Course Notes: Operational Semantics and the Parameterized Aspect Calculus

Curtis Clifton and Gary T. Leavens
Dept. of Computer Science
Iowa State University
226 Atanasoff Hall
Ames, IA 50011-1040 USA
{cclifton,leavens}@cs.iastate.edu

December 5, 2003

1 Motivation

1.1 Review [4, 7]

- Quantification

  Defn. 1.1 (Quantified Statements) *have an effect on many places* in the program

  as opposed to "in the underlying code", which is biased toward the base + aspects model

- Obliviousness

  Defn. 1.2 (Obliviousness) *the execution of cross-cutting code A without any reference to A from the client code that A cross-cuts*

  - semantic interaction
  - without syntactic coupling

- Modular Reasoning

  Understanding a module M based on:

  - the code *in* M,
  - the code *surrounding* M, and
  - the *signature* and *specification* of any modules referred to by that code.

- Behavioral Subtyping Analogy

- Behavioral subtyping in OOP:
  an overriding method must satisfy the specification of the overridden method
- Behavioral subtyping is a *discipline*
  * It places constraints on the subtype programmer
  * It provides the benefit of modular reasoning for clients
- What about AOP?
  Q: Can a language have quantification and obliviousness *and* allow modular reasoning?

  It isn't clear.
  Q: Is there a discipline like behavioral subtyping that would allow modular reasoning in aspect-oriented programming languages? in AspectJ?

1.2 Spectators and Assistants [3]

- Assistants
  - can change the behavior of advised code
  - must be explicitly accepted by either
    * the module containing the advised join points,
      (all clients see the effects)
    * or a client of that module
      (only that client sees the effects)
- Spectators

  Defn. 1.3 *A* spectator *is an aspect that "does not change the behavior of any other module."*

  Q: What might that mean? What is "spectator-ness"?

  - Safety and Liveness [10]

    Defn. 1.4 *A* safety property *says that nothing bad happens*

    Defn. 1.5 *A* liveness property *says that eventually something good happens*

    * Before-advice that immediately went into an infinite loop would be *safe* but not *live*
    * Before-advice that deleted all the files on your hard drive and then proceeded to the original method would be *live* but not *safe*
  - Spectators and Safety
    Some possible interpretations:
    * A spectator cannot modify any state but its own

    - A spectator cannot violate the specification of advised modules

  Q: Is it that simple? Are there any problems with these notions?

  What about I/O?
  Can we modularly find all the advised modules? What about quantification?

- Spectators and Liveness
  Goal: Spectators must always allow the advised method to execute with its original arguments and must return the result unchanged.
  Q: Is this decidable?

  No! by reduction from the halting problem.

  What if we:

    - Restrict control flow constructs in spectator advice to make the problem decidable?

      Q: What constructs could we allow? loops? method calls? mathematical expressions?

    - Run spectators in a separate thread?

      Q: What if advice isn't finished before advised method is called again?

    - Approximate by prohibiting spectators from using around-advice or throwing checked exceptions?

- Do you buy it? (Direct discussion towards needing formal proof.)
  - Which of these notions of "spectator-ness" could be statically enforced? All but the specification safety property (and perhaps that could be if the specfications were sufficiently restricted).
  - Do spectators and assistants provide modular reasoning? How do we know?
  - Can we implement reasonable aspect-oriented programs under these restrictions?

1.3 Why formal semantics?

Defn. 1.6 *A* formal semantics *is a mathematically complete description of a programming language*

- Makes proofs about language properties tractable
- *Lingua franca* of programming language researchers

1.4 Why core calculi?

Defn. 1.7 *A* core calculus *is a programming language* *stripped of all but its essential elements*

Q: What is "essential"? Depends on the problem
A core calculus:

- Eliminates "noise"
- Makes construction of complete formal semantics tractable
- Can be used to define user-level languages
- Examples
  - λ calculus and Haskell
  - Object calculus and Smalltalk
  - Parameterized aspect calculus and AspectJ?

2 Introduction to Formal Semantics

2.1 Kinds of Formal Semantics

Example: the semantics of a while loop

- Denotational [9]
  - Strength: proving properties about the language
  - Map values in language to mathematical entities, like $\{\mathsf{T}, \mathsf{F}\}$ or the natural numbers
  - Model operations in language as mathematical operations, like $\wedge$, $\neg$, or $+$
  - Example:

$$[\![\text{while } E \text{ do } C;]\!]_s = w(s), \text{ where } w(s) = \mathit{if}([\![E]\!]_s, w([\![C]\!]_s), s)$$

    s is the state, typically a mapping from variables to values
    Read double brackets as "the meaning of foo in the state s".
    w is recursive

    $[\![]\!]_s$ is overloaded:
    * $[\![E]\!]_s$: *boolean*
    * $[\![C]\!]_s$: *state*
    * **Q:** what is the type of the if function?
      A: $\mathit{if}: \mathit{Boolean} \times \mathit{State} \times \mathit{State} \rightarrow \mathit{State}$

- Axiomatic [2]

- Strength: proving properties about actual programs
- Map values in language to mathematical entities
- Describe operations using logical assertions, for example pre- and post-conditions and loop invariants
- Uses *Hoare triples*: $\{P\}C\{Q\}$
  * P is a pre-condition
  * Q is a post-condition
  * For two states s and s' we write:

  $$(s, s') \vDash \{P\}C\{Q\} \text{ iff } [\![P]\!]_s \wedge ([\![C]\!]_s = s') \wedge [\![Q]\!]_{s'}$$

  We say "the Hoare triple $\{P\}C\{Q\}$ is valid for the pair of states (s, s')."

- Example:

  $$\frac{\{I \wedge E\}C\{I\}}{\{I\}\text{while } E \text{ do } C;\{I \wedge \neg E\}}$$

  I is the loop invariant

  Typically the rule used is actually:

  $$\frac{P \Rightarrow I \qquad \{I \wedge E\}C\{I\} \qquad (I \wedge \neg E) \Rightarrow Q}{\{P\}\text{while } E \text{ do } C;\{Q\}}$$

- Operational
  - Strength: clarity, guides implementation, proving behavioral properties of the language
  - Values in language represent themselves (typically)
  - Operations are described by *rewrite rules* that *reduce* a term to a new term, given that a set of premises is satisfied.

    General form:

    $$\frac{premise_1 \qquad \ldots \qquad premise_n}{Env \vdash a \rightsquigarrow b}$$

    Env is an environment
    a and b might be terms, or might be sequences describing the state of some virtual machine (e.g., term + state)

– Two sorts of operational semantics

  * Small Step: a sub-term of a is replaced with a new sub-term to form b rules chain horizontally
    Example:
    The semantics of the if statement is:

$$\frac{}{\vdash \textsf{if true then } C_0 \textsf{ else } C_1 \cdot s \to C_0 \cdot s} \qquad \frac{}{\vdash \textsf{if false then } C_0 \textsf{ else } C_1 \cdot s \to C_1 \cdot s}$$

$$\frac{\vdash E \cdot s \to E' \cdot s'}{\vdash \textsf{if } E \textsf{ then } C_0 \textsf{ else } C_1 \cdot s \to \textsf{if } E' \textsf{ then } C_0 \textsf{ else } C_1 \cdot s'}$$

    and the semantics of statement sequencing is:

$$\frac{}{\vdash \textsf{skip}; C_1 \cdot s \to C_1 \cdot s} \qquad \frac{\vdash C_0 \cdot s \to C_0' \cdot s'}{\vdash C_0; C_1 \cdot s \to C_0'; C_1 \cdot s'}$$

    Using these, the semantics of the while statement is [8]:

$$\frac{}{\vdash \textsf{while } E \textsf{ do } C; \cdot s \to \textsf{if } E \textsf{ then } C; \textsf{while } E \textsf{ do } C; \textsf{ else skip} \cdot s}$$

    Reduction terminates with $\langle \textsf{skip}, s\rangle$.

  * Big Step (a.k.a. "natural"): a is reduced to a value in one (big) step rules stack vertically
    Sometimes when people (e.g., Abadi and Cardelli) say "operational semantics", they mean big step
    Example:

$$\frac{\vdash E \cdot s \rightsquigarrow \textsf{false} \cdot s'}{\vdash \textsf{while } E \textsf{ do } C; \cdot s \rightsquigarrow s'}$$

$$\frac{\vdash E \cdot s \rightsquigarrow \textsf{true} \cdot s_e \qquad \vdash C \cdot s_e \rightsquigarrow s' \qquad \vdash \textsf{while } E \textsf{ do } C; \cdot s' \rightsquigarrow s''}{\vdash \textsf{while } E \textsf{ do } C; \cdot s \rightsquigarrow s''}$$

    The result of reducing a statement is just the state.
    Reducing an expression just yields a value, assuming expressions cannot have side effects.

- Other kinds of formal semantics
  – Labelled transition systems (enhancement of small step op sem)
  – Chemical semantics

2.2 Operational semantics for the λ calculus

- Small step semantics (review, but in Abadi and Cardelli format)
  - Rules
    - Top-level, one-step reduction omitting alpha and eta rules

$$\frac{\beta}{\vdash ((\lambda x.e)\ e') \rightarrowtail e\{\!\{x \leftarrow e'\}\!\}}$$

A&C substitution style, and sometimes the x is omitted

    - One-step reduction

Defn. 2.1 *A* context $\mathcal{C}[\![-]\!]$ *is a term with* a single hole.
$\mathcal{C}[\![e]\!]$ *represents the result of* filling the hole with the term e (possibly capturing free variables of e).

$$\frac{\vdash e \rightarrowtail e' \qquad \mathcal{C}[\![-]\!] \text{ is any context}}{\vdash \mathcal{C}[\![e]\!] \rightarrow \mathcal{C}[\![e']\!]}$$

    - Many-step reduction
$\twoheadrightarrow$ is the reflexive transitive closure of $\rightarrow$
    - Example

$$\frac{\dfrac{}{\vdash ((\lambda z.z)\ 2) \rightarrowtail 2} \qquad \mathcal{C}[\![-]\!] = ((\lambda y.3)\ -)}{\vdash ((\lambda y.3)\ ((\lambda z.z)\ 2)) \rightarrow ((\lambda y.3)\ 2)} \qquad \frac{\dfrac{}{\vdash ((\lambda y.3)\ 2) \rightarrowtail 3} \qquad \mathcal{C}[\![-]\!] = -}{\vdash ((\lambda y.3)\ 2) \rightarrow 3}$$

Rules chain horizontally

  - Non-deterministic:

$$((\lambda y.3)\ ((\lambda x.(x\ x))\ (\lambda x.(x\ x))))$$

Can be made deterministic by restricting the shape of contexts.

    - Normal order: $\mathcal{C}[\![-]\!] ::= - \ \mid\ (\mathcal{C}[\![-]\!]\ e)$
    - Applicative order?

Need a notion of values
$\mathcal{C}[\![-]\!] ::= - \ \mid\ (v\ \mathcal{C}[\![-]\!])\ \mid\ (\mathcal{C}[\![-]\!]\ e)$
Need to restrict the β rule to reduce only terms of the form $((\lambda x.e)\ v)$.

- Big step semantics
  - Judgment: $\vdash e \rightsquigarrow v$
    The term e reduces to the value v
  - Values
    * λ terms, $(\lambda x.e)$
    * free variables
  - Rules

$$\beta\ \frac{\vdash e\{\!\{x \leftarrow e'\}\!\} \rightsquigarrow v}{\vdash ((\lambda x.e)\ e') \rightsquigarrow v} \qquad \textsc{Rator}\ \frac{\vdash e \rightsquigarrow v' \qquad \vdash (v\ e') \rightsquigarrow v \qquad e \text{ is not a value}}{\vdash (e\ e') \rightsquigarrow v} \qquad \textsc{Val}\ \frac{}{\vdash v \rightsquigarrow v}$$

Q: Do these rules describe applicative order? normal order? some other order? normal order

Homework: Give the big step semantics for applicative order reduction. E.C.: implement interpreter based on big step semantics

  - Examples

$$\frac{\dfrac{}{\vdash 3 \rightsquigarrow 3}\ \textsc{Value}}{\vdash ((\lambda \mathsf{y}.3)\ ((\lambda \mathsf{z}.\mathsf{z})\ 2)) \rightsquigarrow 3}\ \beta$$

Let them work out this one:

$$\frac{\dfrac{\dfrac{}{\vdash (\lambda \mathsf{y}.3) \rightsquigarrow (\lambda \mathsf{y}.3)}\ \textsc{Value}}{\vdash ((\lambda \mathsf{x}.\mathsf{x})\ (\lambda \mathsf{y}.3)) \rightsquigarrow (\lambda \mathsf{y}.3)}\ \beta \qquad \dfrac{\dfrac{}{\vdash 3 \rightsquigarrow 3}\ \textsc{Value}}{\vdash ((\lambda \mathsf{y}.3)\ ((\lambda \mathsf{z}.\mathsf{z})\ 2)) \rightsquigarrow 3}\ \beta}{\vdash (((\lambda \mathsf{x}.\mathsf{x})\ (\lambda \mathsf{y}.3))\ ((\lambda \mathsf{z}.\mathsf{z})\ 2)) \rightsquigarrow 3}\ \textsc{Rator}$$

  - **Q:** Is this semantics deterministic?
    Yes, because only one rule is applicable to any term.

- Abadi and Cardelli Proof Style [1, pp. 79–80]

$\quad\quad Judg_2$	(RULE 2)
$\quad Judg_3$	(RULE 3)
$\quad\quad Judg_4$	REASON
$\quad Judg_5$	(RULE 5)
$Judg_6$	

Example:

$$
\begin{array}{ll}
\quad\quad \vdash (\lambda y.3) \leadsto (\lambda y.3) & \textsc{Value} \\
\quad \vdash ((\lambda x.x)\ (\lambda y.3)) \leadsto (\lambda y.3) & \beta \\
\quad\quad \vdash 3 \leadsto 3 & \textsc{Value} \\
\quad \vdash ((\lambda y.3)\ ((\lambda z.z)\ 2)) \leadsto 3 & \beta \\
\vdash (((\lambda x.x)\ (\lambda y.3))\ ((\lambda z.z)\ 2)) \leadsto 3 & \textsc{Rator}
\end{array}
$$

2.3 Untyped Object Calculus, ς

- Syntax

$$
\begin{array}{rrcl}
\text{variables} & x & \in & \mathit{Vars} \\
\text{labels} & l & \in & \mathit{Labels} \\
\text{terms} & a, b, c & ::= & x \\
 & & | & [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}] \\
 & & | & a.l \\
 & & | & a.l \Leftarrow \varsigma(x)b
\end{array}
$$

- Big step semantics (omitting small step semantics due to limited time)

  Homework: Implement a stack object using the object calculus

  - Object: a *set* of pairs of *labels* and *methods*

$$
\textsc{Red Object}\quad \frac{}{\vdash [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}] \leadsto [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}]}
$$

    Example: [pos=ς(x)x.n, n=ς(x)2], where 2 is shorthand for an object that represents the natural number 2.

  - Method Selection: reduces the body of the *named method*, substituting object for the *self parameter*

$$
\textsc{Red Select}\quad \frac{\vdash a \leadsto [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}] \qquad \vdash b_j\{\!\{x_j \leftarrow [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}]\}\!\} \leadsto v \qquad j \in I}{\vdash a.l_j \leadsto v}
$$

    Example: [pos=ς(x)x.n, n=ς(x)2].pos

$$
\begin{array}{ll}
\quad \vdash [\mathsf{pos} = \varsigma(\mathsf{x})\mathsf{x.n}, \mathsf{n} = \varsigma(\mathsf{x})2] \leadsto [\mathsf{pos} = \varsigma(\mathsf{x})\mathsf{x.n}, \mathsf{n} = \varsigma(\mathsf{x})2] & \textsc{Red Object} \\
\quad \mathsf{pos} \in \{\mathsf{pos}, \mathsf{n}\} & \\
\quad\quad \vdash [\mathsf{pos} = \varsigma(\mathsf{x})\mathsf{x.n}, \mathsf{n} = \varsigma(\mathsf{x})2] \leadsto [\mathsf{pos} = \varsigma(\mathsf{x})\mathsf{x.n}, \mathsf{n} = \varsigma(\mathsf{x})2] & \textsc{Red Object} \\
\quad\quad \mathsf{n} \in \{\mathsf{pos}, \mathsf{n}\} & \\
\quad\quad \vdash 2 \leadsto 2 & \textsc{Red Object} \\
\quad \vdash [\mathsf{pos} = \varsigma(\mathsf{x})\mathsf{x.n}, \mathsf{n} = \varsigma(\mathsf{x})2].\mathsf{n} \leadsto 2 & \textsc{Red Select} \\
\vdash [\mathsf{pos} = \varsigma(\mathsf{x})\mathsf{x.n}, \mathsf{n} = \varsigma(\mathsf{x})2].\mathsf{pos} \leadsto 2 & \textsc{Red Select}
\end{array}
$$

– Method update: generates a *new* object, with the given method replacing the named method

$$\frac{\vdash a \rightsquigarrow [\overline{l_i = \varsigma(x_i)b_i}^{i \in I}] \qquad j \in I}{\vdash a.l_j \Leftarrow \varsigma(x)b \rightsquigarrow [l_j = \varsigma(x)b, \overline{l_i = \varsigma(x_i)b_i}^{i \in I \setminus j}]} \text{ RED UPDATE}$$

Q: What's the result of reducing this term: [pos=ς(x)x.n, n=ς(x)2].n ⇐ς(x)3

A: [pos=ς(x)x.n, n=ς(x)3]

Q: What about this one: [pos=ς(x)x.n, n=ς(x)2].pos ⇐ς(x)x.n.succ

A: [pos=ς(x)x.n.succ, n=ς(x)2]

Q: What happens if we select pos on the result?

A: 3, assuming 2.succ $\rightsquigarrow$ 3

- Syntactic sugar

  – Fields: methods in which the self parameter does not appear free
  [pos=ς(x).n, n=2] desugars to [pos=ς(x).n, n=ς(y)2] where y is not free in 2
  [pos=ς(x).n, n=2].n := 3 desugars to [pos=ς(x).n, n=3]

  – Lambda expressions
  Can translate untyped λ calculus into the ς calculus.
  Let $\langle\!\langle\rangle\!\rangle$ map λ calculus to ς calculus as follows:

$$\begin{array}{rcl} \langle\!\langle x \rangle\!\rangle & = & x \\ \langle\!\langle (e_1\ e_2) \rangle\!\rangle & = & (\langle\!\langle e_1 \rangle\!\rangle.arg{:=}\langle\!\langle e_2 \rangle\!\rangle).val \\ \langle\!\langle (\lambda x.e) \rangle\!\rangle & = & [arg = 0, val = \varsigma(s)\langle\!\langle e \rangle\!\rangle\{\!\{x \leftarrow s.arg\}\!\}] \end{array}$$

  Homework: Translate some lambda calculus expressions and reduce them in the object calculus

3 Parameterized Aspect Calculus, ς_{asp} [5, 6]

3.1 Changes vs. the object calculus

Object calculus plus aspects plus constants

- Join point abstraction

  – Each reduction step triggers a search for advice

  – Search uses a four-part abstraction of the reduction step

    * *Reduction kind*, ρ, one of {VAL, IVK, UPD}
    * *Evaluation context*, $\mathcal{K}$, represents the call stack
    * *Target signature*, represents the "shape" of the target of the operation
      · either the set of labels in the target object, or
      · the name of a constant
    * Invocation or update *message*

    - either a label, or
    - a functional constant
  - The search semantics is specified by a *point cut description language*, or *PCDL*
    * PCDL is a parameter to the calculus, various PCDL may be used
      Q: How might this be useful?

      A: can easily experiment with different PCDL
      A: can restrict the set of join points that might be matched

      Q: What problems might this cause?

      A: might make the semantics more complex
      A: possible that complexity is hidden in the PCDL, making the core calculus "less core"

    * PCDL consists of two parts:
      - Point cut description syntax, $\mathcal{C}$
      - Advice matching function, $match$

- Syntax of ς_{asp}
  - All object calculus terms
  - Constants

$$d \in Consts \qquad f \in FConsts \qquad \text{terms} \quad a,b,c \; ::= \; \ldots \mid d \mid a.f$$

Constants are things like natural numbers
Functional constants are operations like successor
The primary reason for introducing constants is to simplify examples, going forward they may be eliminated–discuss this if time allows

  - Advice

$$pcd \in \mathcal{C} \qquad \begin{array}{rrcl} \text{programs} & \mathcal{P} & ::= & a \otimes \overrightarrow{\mathcal{A}} \\ \text{advice} & \mathcal{A} & ::= & pcd \triangleright \varsigma(\overrightarrow{y})b \end{array}$$

A program consists of a base term (think "main") and a sequence of advice
Advice maps a point cut description to a "naked method", define naked method

– Proceeding

$$
\begin{array}{rrcl}
\text{terms} & a,b,c & ::= & \ldots \\
 & & | & \mathsf{proceed}_{\mathrm{VAL}}() \\
 & & | & \mathsf{proceed}_{\mathrm{IVK}}(a) \\
 & & | & \mathsf{proceed}_{\mathrm{UPD}}(a, \varsigma(x)b) \\
 & & | & \pi \\
\text{proceed closures} & \pi & ::= & \Pi_{\mathrm{VAL}}\{\!|B, v|\!\}() \\
 & & | & \Pi_{\mathrm{IVK}}\{\!|B, S, k|\!\}(a) \\
 & & | & \Pi_{\mathrm{UPD}}\{\!|B, k|\!\}(a, \varsigma(x)b)
\end{array}
$$

Advice can contain proceed terms
proceed terms are converted to proceed closures during advice lookup
User programs cannot contain proceed closures

- Semantics
  - Changes
    - Object calculus reduction rules are changed to add advice lookup
    - Rules are added for:
      - Constants
      - Object calculus terms to which advice applies
      - Proceeding
  - Helper functions
    - Advice lookup

$$advFor_{\boldsymbol{M}}(jp, \bullet) = \bullet$$

$$advFor_{\boldsymbol{M}}(jp, (pcd \triangleright \varsigma(\overrightarrow{y})b) + \overrightarrow{\mathcal{A}}) = match(pcd \triangleright \varsigma(\overrightarrow{y})b, jp) + advFor_{\boldsymbol{M}}(jp, \overrightarrow{\mathcal{A}})$$

Returns a list of naked methods
Invokes PCDL's $match$ function for each piece of advice

* Proceed closure

$$close_{\text{VAL}}(\mathsf{proceed}_{\text{VAL}}(), \{\!|B, v|\!\}) = \Pi_{\text{VAL}}\{\!|B, v|\!\}()$$

$$\begin{array}{l} close_{\text{IVK}}(\mathsf{proceed}_{\text{IVK}}(a), \{\!|B, S, k|\!\}) = \\ \qquad \Pi_{\text{IVK}}\{\!|B, S, k|\!\}(close_{\text{IVK}}(a, \{\!|B, S, k|\!\})) \end{array}$$

$$\begin{array}{l} close_{\text{UPD}}(\mathsf{proceed}_{\text{UPD}}(a, \varsigma(x)b), \{\!|B, k|\!\}) = \\ \quad \Pi_{\text{UPD}}\{\!|B, k|\!\}(close_{\text{UPD}}(a, \{\!|B, k|\!\}), \varsigma(x)close_{\text{UPD}}(b, \{\!|B, k|\!\})) \end{array}$$

Takes proceed terms in advice and converts them to proceed closures, squirreling away any information needed for proceeding.
These are the most interesting definitions, the others just recurse to sub-terms.

– Objects and Basic Constants

$$\text{values} \quad v \quad ::= \quad d \ \mid \ [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}]$$

$$\textsc{Red Val 0}\quad \frac{\mathcal{K} \vdash_{\boldsymbol{M},\vec{\mathcal{A}}} \diamond \qquad advFor_{\boldsymbol{M}}(\langle \text{VAL}, \mathcal{K}, sig(v), \epsilon\rangle, \vec{\mathcal{A}}) = \bullet}{\mathcal{K} \vdash_{\boldsymbol{M},\vec{\mathcal{A}}} v \rightsquigarrow v}$$

$$\textsc{Red Val 1}\quad \frac{\begin{array}{c}\mathcal{K} \vdash_{\boldsymbol{M},\vec{\mathcal{A}}} \diamond \qquad advFor_{\boldsymbol{M}}(\langle \text{VAL}, \mathcal{K}, sig(v), \epsilon\rangle, \vec{\mathcal{A}}) = \varsigma()b + B \\ close_{\text{VAL}}(b, \{\!|B, v|\!\}) = b' \qquad \mathsf{va} \cdot \mathcal{K} \vdash_{\boldsymbol{M},\vec{\mathcal{A}}} b' \rightsquigarrow v'\end{array}}{\mathcal{K} \vdash_{\boldsymbol{M},\vec{\mathcal{A}}} v \rightsquigarrow v'}$$

Q: What, in plain English, is the meaning of these two rules?

Things to note:

* subscripts on the turnstile
* wellformedness premise
* Red Val 0 correspondence to Red Object
* advice lookup
    * join point abstraction

  · Required shape of result in RED VAL 1

* proceed closure, and information stored
* evaluation context in last premise of RED VAL 1

– Method Selection

RED SEL 0 (where $o \triangleq [\overline{l_i = \varsigma(x_i)b_i}^{\,i\in I}]$)

$$\frac{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} a \rightsquigarrow o \qquad l_j \in \overline{l_i}^{\,i\in I} \qquad \mathit{advFor}_{\boldsymbol{M}}(\langle \text{IVK}, \mathcal{K}, \overline{l_i}^{\,i\in I}, l_j\rangle, \vec{\mathcal{A}}) = \bullet \qquad \mathsf{ib}(\overline{l_i}^{\,i\in I}, l_j)\cdot\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} b_j\{\!\{x_j \leftarrow o\}\!\} \rightsquigarrow v}{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} a.l_j \rightsquigarrow v}$$

RED SEL 1 (where $o \triangleq [\overline{l_i = \varsigma(x_i)b_i}^{\,i\in I}]$)

$$\frac{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} a \rightsquigarrow o \qquad l_j \in \overline{l_i}^{\,i\in I} \qquad \mathit{advFor}_{\boldsymbol{M}}(\langle \text{IVK}, \mathcal{K}, \overline{l_i}^{\,i\in I}, l_j\rangle, \vec{\mathcal{A}}) = \varsigma(y)b + B \qquad \mathit{close}_{\text{IVK}}(b, \{\!\!\{(B + \varsigma(x_j)b_j), \overline{l_i}^{\,i\in I}, l_j\}\!\!\}) = b' \qquad \mathsf{ia}\cdot\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} b'\{\!\{y \leftarrow o\}\!\} \rightsquigarrow v}{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} a.l_j \rightsquigarrow v}$$

Q: What, in plain English, is the meaning of these two rules?
Q: Where does the final value come from?

Things to note:

* correspondence of RED SEL 0 and RED SELECT
* join point abstraction
* shape of returned advice
* information stored in proceed closure
* evaluation context differences

– Functional Constant Application

$\delta(f, v')$ means "apply the functional constant f to the value v'. δ is intentionally underspecified, since we don't say what the basic and functional constants are. Suppose $FConsts = \{\mathsf{succ}\}$ and $Consts$ is the natural numbers: $\delta(\mathsf{succ}, 3) = 4$.

RED FCONST 0

$$\frac{\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a \rightsquigarrow v' \qquad \mathit{advFor}_{\boldsymbol{M}}(\langle \textsc{Ivk}, \mathcal{K}, \mathit{sig}(v'), f\rangle, \overrightarrow{\mathcal{A}}) = \bullet \qquad \mathsf{ib}(\mathit{sig}(v'), f) \cdot \mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} \delta(f, v') \rightsquigarrow v}{\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a.f \rightsquigarrow v}$$

RED FCONST 1

$$\frac{\begin{array}{c}\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a \rightsquigarrow v' \qquad \mathit{advFor}_{\boldsymbol{M}}(\langle \textsc{Ivk}, \mathcal{K}, \mathit{sig}(v'), f\rangle, \overrightarrow{\mathcal{A}}) = \varsigma(y)b + B \\ \mathit{close}_{\textsc{Ivk}}(b, \{\!|B, \mathit{sig}(v'), f|\!\}) = b' \qquad \mathsf{ia} \cdot \mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} b'\{\!\{y \leftarrow v'\}\!\} \rightsquigarrow v\end{array}}{\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a.f \rightsquigarrow v}$$

Q: What is the meaning of these two rules?

Things to note:

- **Q:** Aren't these rules non-deterministic given the selection rules? Not if $\mathit{FConsts} \cup \mathit{Labels} = \varnothing$
- **Q:** How do these rules differ from the selection rules?

  No label presence test
  Join point abstraction uses sig function
  The 0 rule uses δ function

– Method Update

RED UPD 0 (where $o \triangleq [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}]$)

$$\frac{\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a \rightsquigarrow o \qquad l_j \in \overline{l_i}^{\,i \in I} \qquad \mathit{advFor}_{\boldsymbol{M}}(\langle \textsc{Upd}, \mathcal{K}, \overline{l_i}^{\,i \in I}, l_j\rangle, \overrightarrow{\mathcal{A}}) = \bullet}{\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a.l_j \Leftarrow \varsigma(x)b \rightsquigarrow [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I \setminus \{j\}}, l_j = \varsigma(x)b]}$$

RED UPD 1 (where $o \triangleq [\overline{l_i = \varsigma(x_i)b_i}^{\,i \in I}]$)

$$\frac{\begin{array}{c}\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a \rightsquigarrow o \qquad \mathit{advFor}_{\boldsymbol{M}}(\langle \textsc{Upd}, \mathcal{K}, \overline{l_i}^{\,i \in I}, l_j\rangle, \overrightarrow{\mathcal{A}}) = \varsigma(\mathit{targ}, \mathit{rval})b' + B \\ \mathit{close}_{\textsc{Upd}}(b', \{\!|B, l_j|\!\}) = b'' \qquad \mathsf{ua} \cdot \mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} b''\{\!\{\mathit{rval} \hookleftarrow b\{\!\{x \leftarrow \mathit{targ}\}\!\}\}\!\}_{\mathit{targ}}\{\!\{\mathit{targ} \leftarrow o\}\!\} \rightsquigarrow v\end{array}}{\mathcal{K} \vdash_{\boldsymbol{M},\overrightarrow{\mathcal{A}}} a.l_j \Leftarrow \varsigma(x)b \rightsquigarrow v}$$

Things to note:

* Correspondence of Red Upd 0 and Red Update
* Evaluation context in Red Upd 1
* Data used for proceed closure
* Shape of returned advice: *two* parameters
  · $targ$, corresponds to the target object, o, of the update operation.
  · $rval$, corresponds to the body, b, of the update's r-value.
* *two* kinds of substitution
  · $b\{\!\{x \leftarrow c\}\!\}$ is normal capture-avoiding substitution
  Key rules: the rest just recurse over the grammar

$$
\begin{array}{rcl}
(\varsigma(y)b)\{\!\{x \leftarrow c\}\!\} & \triangleq & \varsigma(y')(b\{\!\{y \leftarrow y'\}\!\}\{\!\{x \leftarrow c\}\!\}) \\
 & & \text{where } y' \notin FV(\varsigma(y)b) \cup FV(c) \cup \{x\} \\
x\{\!\{x \leftarrow c\}\!\} & \triangleq & c \\
y\{\!\{x \leftarrow c\}\!\} & \triangleq & y \qquad\qquad\qquad\qquad \text{if } x \neq y
\end{array}
$$

  · $b''\{\!\{x \hookleftarrow c\}\!\}_z$ says: in b'' replace all free occurances of x with c, capturing any free occurances of z in c
  Key rules: varref is same as above, the rest just recurse over the grammar

$$
\begin{array}{rcll}
(\varsigma(z)b)\{\!\{x \hookleftarrow c\}\!\}_z & \triangleq & \varsigma(z)(\{\!\{x \hookleftarrow c\}\!\}_z) & \text{no renaming} \\
(\varsigma(y)b)\{\!\{x \hookleftarrow c\}\!\}_z & \triangleq & \varsigma(y')(b\{\!\{y \leftarrow y'\}\!\}\{\!\{x \hookleftarrow c\}\!\}_z) & \text{renaming} \\
 & & \text{if } y \neq z \text{, where } y' \notin FV(\varsigma(y)b) \cup FV(c) \cup \{x\} &
\end{array}
$$

  Q: Which of these rules does the capturing?
  A: the first
* Why two kinds of substitution? solicit ideas
  · $b\{\!\{x \leftarrow targ\}\!\}$: renames the self parameter in the body, b, of the original r-value
  · $targ$-capturing substitution for $rval$ in the advice body, b'', lets advice author:
  capture occurrences of the self-parameter, by placing $rval$ under a $\varsigma(targ)$ binder
  or
  not capture occurrences of the self-parameter, by not placing $rval$ under a binder or by placing it under a non-$targ$ binder
* Examples:

[n=ς(y)0, pos=ς(p)p.n].pos ⇐ ς(x)x.n.succ

  · In the absence of advice, this would reduce to:

[n=ς(y)0, pos=ς(x)x.n.succ]

  Q: What happens if we update n to 2 in this object and then select pos?
  A: We get back 3.

· Advice designed to avoid capture: targ does not appear bound in b''

$$\varsigma(\text{targ,rval})\text{proceed}_{\text{UPD}}(\text{targ}, \varsigma(\text{z})\text{rval})$$

fixes the value of the pos method to the result of evaluating the new method body, x.n.succ, substituting the original target object for x:

Assuming no other advice:

$$b'' = \Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ}, \varsigma(\text{z})\text{rval})$$

Underbars indicate target of next substitution

$$\begin{aligned}
&\Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ}, \varsigma(\text{z})\text{rval})\{\!|\text{rval} \hookleftarrow \underline{\text{x.n.succ}}\{\!|\text{x} \leftarrow \text{targ}|\!\}|\!\}_{\text{targ}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\!|\text{targ} \leftarrow [\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}]|\!\} \\
&= \underline{\Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ}, \varsigma(\text{z})\text{rval})}\{\!|\text{rval} \hookleftarrow \text{targ.n.succ}|\!\}_{\text{targ}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\!|\text{targ} \leftarrow [\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}]|\!\} \\
&= \underline{\Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ}, \varsigma(\text{z})\text{targ.n.succ})}\{\!|\text{targ} \leftarrow [\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}]|\!\} \\
&= \Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}([\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}], \varsigma(\text{z})[\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}].\text{n.succ})
\end{aligned}$$

The last term will reduce to:

$$[\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{z})[\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}].\text{n.succ}]$$

Q: What happens if we update n to 2 in this object and then select pos?
A: We get back 1!

· Advice designed to capture: because rval appears under a targ binder

$$\varsigma(\text{targ,rval})\text{proceed}_{\text{UPD}}(\text{targ},\varsigma(\text{targ})\text{rval.succ})$$

uses the body of the update's r-value without causing it to be reduced

Assuming no other advice was found in the advice lookup, then after closing the $\text{proceed}_{\text{UPD}}$ sub-term, the substitutions for this advice are:

$$\begin{aligned}
&\Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ},\varsigma(\text{targ})\text{rval.succ})\ \{\!|\text{rval} \hookleftarrow \underline{\text{x.n.succ}}\{\!|\text{x} \leftarrow \text{targ}|\!\}|\!\}_{\text{targ}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\!|\text{targ} \leftarrow [\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}]|\!\} \\
&= \underline{\Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ},\varsigma(\text{targ})\text{rval.succ})}\{\!|\text{rval} \hookleftarrow \text{targ.n.succ}|\!\}_{\text{targ}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\!|\text{targ} \leftarrow [\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}]|\!\} \\
&= \underline{\Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}(\text{targ},\varsigma(\text{targ})\text{targ.n.succ.succ})}\ \text{capture!} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\!|\text{targ} \leftarrow [\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}]|\!\} \\
&= \Pi_{\text{UPD}}\{\!|\bullet, \text{pos}|\!\}([\text{n}=\varsigma(\text{y})0, \text{pos}=\varsigma(\text{p})\text{p.n}], \varsigma(\text{targ})\qquad\qquad \text{targ.n.succ.succ})
\end{aligned}$$

The last targ is not free and so isn't replaced.
(Those last two targ's should really be renamed, but this is alpha equivalent.)

This term will reduce to:

[n=ς(y)0, pos=ς(targ)targ.n.succ.succ]

Q: What happens if we update n to 2 in this object and then select pos?
A: We get back 4!

- Proceeding
  - General ideas:
    - Two rules for each kind of advice one for proceeding to lower precedence advice, one for proceeding to original operation
    - Rules are very similar to the regular operations, *except* ...
    - No additional advice lookup subsequent advice and original operation are taken from the proceed closure
    - Proceed closure formed lazily
  - Proceeding from Value Advice

$$\textsc{Red VPrcd 0}\quad \frac{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} \diamond}{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} \Pi_{\mathrm{VAL}}\{\!|\bullet, v|\!\}() \rightsquigarrow v}$$

$$\textsc{Red VPrcd 1}\quad \frac{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} \diamond \qquad close_{\mathrm{VAL}}(b, \{\!|B, v|\!\}) = b' \qquad \mathsf{va} \cdot \mathcal{K} \vdash_{M,\vec{\mathcal{A}}} b' \rightsquigarrow v'}{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} \Pi_{\mathrm{VAL}}\{\!|(\varsigma()b + B), v|\!\}() \rightsquigarrow v'}$$

  - Proceeding from Selection Advice

$$\textsc{Red SPrcd 0}\quad \frac{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} a \rightsquigarrow o \qquad \mathsf{ib}(\bar{l}, l) \cdot \mathcal{K} \vdash_{M,\vec{\mathcal{A}}} b\{\!\{y \leftarrow o\}\!\} \rightsquigarrow v}{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} \Pi_{\mathrm{IVK}}\{\!|\varsigma(y)b, \bar{l}, l|\!\}(a) \rightsquigarrow v}$$

$$\textsc{Red SPrcd 1}\quad \frac{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} a \rightsquigarrow o \qquad B \neq \bullet \qquad close_{\mathrm{IVK}}(b, \{\!|B, \bar{l}, l|\!\}) = b' \qquad \mathsf{ia} \cdot \mathcal{K} \vdash_{M,\vec{\mathcal{A}}} b'\{\!\{y \leftarrow o\}\!\} \rightsquigarrow v}{\mathcal{K} \vdash_{M,\vec{\mathcal{A}}} \Pi_{\mathrm{IVK}}\{\!|(\varsigma(y)b + B), \bar{l}, l|\!\}(a) \rightsquigarrow v}$$

Q: Where does the target object in the 0 rule come from?
A: the proceed closure's argument
Q: Where does the method body evaluated in the 0 rule come from?
A: the proceed closure's thunk *not* the target object

* Proceeding from Application Advice

$$\textsc{Red FPrcd 0}\quad \frac{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} a \rightsquigarrow v' \qquad \mathsf{ib}(S,f)\cdot\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} \delta(f,v') \rightsquigarrow v}{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} \Pi_{\mathrm{IVK}}\{\!\!|\bullet, S, f|\!\!\}(a) \rightsquigarrow v}$$

$$\textsc{Red FPrcd 1}\quad \frac{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} a \rightsquigarrow v' \qquad close_{\mathrm{IVK}}(b, \{\!\!|B, S, f|\!\!\}) = b' \qquad \mathsf{ia}\cdot\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} b'\{\!\!\{y \leftarrow v'\}\!\!\} \rightsquigarrow v}{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} \Pi_{\mathrm{IVK}}\{\!\!|(\varsigma(y)b + B), S, f|\!\!\}(a) \rightsquigarrow v}$$

* Proceeding from Update Advice

$$\textsc{Red UPrcd 0}\quad \frac{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} a \rightsquigarrow [\overline{l_i = \varsigma(x_i)b_i}^{\,i\in I}] \qquad l_j \in \overline{l_i}^{\,i\in I}}{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} \Pi_{\mathrm{UPD}}\{\!\!|\bullet, l_j|\!\!\}(a, \varsigma(x)b) \rightsquigarrow [\overline{l_i = \varsigma(x_i)b_i}^{\,i\in I\setminus j}, l_j = \varsigma(x)b]}$$

$$\textsc{Red UPrcd 1}\quad \frac{\begin{array}{c}\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} a \rightsquigarrow o \qquad close_{\mathrm{UPD}}(b', \{\!\!|B, l_j|\!\!\}) = b'' \\ \mathsf{ua}\cdot\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} b''\{\!\!\{rval \hookleftarrow b\{\!\!\{x \leftarrow targ\}\!\!\}\}\!\!\}_{targ}\{\!\!\{targ \leftarrow o\}\!\!\} \rightsquigarrow v\end{array}}{\mathcal{K} \vdash_{M,\overrightarrow{\mathcal{A}}} \Pi_{\mathrm{UPD}}\{\!\!|(\varsigma(targ, rval)b' + B), l_j|\!\!\}(a, \varsigma(x)b) \rightsquigarrow v}$$

References

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, New York, NY, 1996.

[2] G. Baumgartner. Axiomatic semantics, Jul 2000. http://www.cis.ohio-state.edu/~gb/cis755/slides/week4-wednesday.pdf.

[3] C. Clifton and G. T. Leavens. Spectators and assistants: Enabling modular aspect-oriented reasoning. Technical Report 02-10, Iowa State University, Department of Computer Science, Oct. 2002.

[4] C. Clifton and G. T. Leavens. Obliviousness, modular reasoning, and the behavioral subtyping analogy. Technical Report 03-01a, Iowa State University, Department of Computer Science, Mar. 2003.

[5] C. Clifton, G. T. Leavens, and M. Wand. Formal definition of the parameterized aspect calculus. Technical Report 03-12b, Iowa State University, Department of Computer Science, Nov. 2003.

[6] C. Clifton, G. T. Leavens, and M. Wand. Parameterized aspect calculus: A core calculus for the direct study of aspect-oriented languages. Technical Report 03-13, Iowa State University, Department of Computer Science, Oct. 2003. Submitted for publication.

[7] R. E. Filman and D. P. Friedman. Aspect-oriented programming is quantification and obliviousness. In M. Akşit, S. Clarke, T. Elrad, and R. E. Filman, editors, *Aspect-Oriented Software Development*. Addison-Wesley, Reading, MA, to appear.

[8] R. Rugina. Small-step operational semantics, Sep 2002. http://www.cs.cornell.edu/courses/cs611/2002fa/lectures/lec05.ps.

[9] D. A. Schmidt. *The Structure of Typed Programming Languages*. Foundations of Computing Series. MIT Press, Cambridge, Mass., 1994.

[10] F. W. Vaandrager. Safety and liveness, Nov 2003. http://www.cs.kun.nl/~fvaan/PV/SLIDES/liveness.pdf.