from Festschrift for **Peter D. Mosses**

An Action Semantics based on Two Combinators

Joint work with **David A. Schmidt**

Kyung-Goo Doh

Denotational Semantics

- Model: λ-abstraction and application
- Style: direct vs. continuation
- Example: Java (excerpt)

$$
\begin{array}{l}
eval\ [\![\ Name\ (\ Arglist\ )\ ]\!]\ env\ econt\ sto = \\
\quad eval\ [\![\ Arglist\ ]\!]\ env\ econt_1\ sto\ \textbf{where} \\
\quad\quad econt_1 = \lambda(vals, typs, sto_1).\ meth(env, scont, sto_1)\ \textbf{where} \\
\quad\quad\quad sig = getSigs(vals)\ \textbf{and} \\
\quad\quad\quad meth = env.getMethod(\textbf{fst}\ (id[\![Name]\!]\ env), sig)\ \textbf{and} \\
\quad\quad\quad scont = \lambda(env_2, sto_2). \\
\quad\quad\quad\quad econt(env_2[\&returnVal], env_2[\&returnType], sto_2)
\end{array}
$$

Denotational Semantics

- Example: Scheme 5 (excerpt)

$\phi \in \mathtt{F}$	$= \mathtt{L} \times (\mathtt{E}^* \to \mathtt{K} \to \mathtt{C})$	procedure values
$\epsilon \in \mathtt{E}$	$= \mathtt{Q} + \mathtt{H} + \mathtt{R} + \mathtt{E_p} + \mathtt{E_v} + \mathtt{E_s} + \mathtt{M} + \mathtt{F}$	expressed values
$\sigma \in \mathtt{S}$	$= \mathtt{L} \to (\mathtt{E} \times \mathtt{T})$	stores
$\rho \in \mathtt{U}$	$= \mathrm{Ide} \to \mathtt{L}$	environments
$\theta \in \mathtt{C}$	$= \mathtt{S} \to \mathtt{A}$	command continuations
$\kappa \in \mathtt{K}$	$= \mathtt{E}^* \to \mathtt{C}$	expression continuations
$\mathtt{A}$		answers
$\mathtt{X}$		errors

7.2.3. Semantic functions

$$\mathcal{K} : \mathrm{Con} \to \mathtt{E}$$
$$\mathcal{E} : \mathrm{Exp} \to \mathtt{U} \to \mathtt{K} \to \mathtt{C}$$
$$\mathcal{E}^* : \mathrm{Exp}^* \to \mathtt{U} \to \mathtt{K} \to \mathtt{C}$$
$$\mathcal{C} : \mathrm{Com}^* \to \mathtt{U} \to \mathtt{C} \to \mathtt{C}$$

$$\begin{array}{l}
\mathcal{E}[\![(\mathrm{E}_0\ \mathrm{E}^*)]\!] = \\
\quad \lambda\rho\kappa\,.\,\mathcal{E}^*(\mathit{permute}(\langle \mathrm{E}_0\rangle\ \S\ \mathrm{E}^*)) \\
\qquad \rho \\
\qquad (\lambda\epsilon^*\,.\,((\lambda\epsilon^*\,.\,\mathit{applicate}\,(\epsilon^* \downarrow 1)\,(\epsilon^* \dagger 1)\,\kappa) \\
\qquad\qquad (\mathit{unpermute}\,\epsilon^*)))
\end{array}$$

$$\begin{array}{l}
\mathcal{E}[\![(\mathtt{lambda}\ (\mathrm{I}^*)\ \Gamma^*\ \mathrm{E}_0)]\!] = \\
\quad \lambda\rho\kappa\,.\,\lambda\sigma\,. \\
\quad\quad \mathit{new}\,\sigma \in \mathtt{L} \to \\
\quad\quad\quad \mathit{send}\,(\langle \mathit{new}\,\sigma \,|\, \mathtt{L}, \\
\quad\quad\quad\quad \lambda\epsilon^*\kappa'\,.\,\#\epsilon^* = \#\mathrm{I}^* \to \\
\quad\quad\quad\quad\quad \mathit{tievals}(\lambda\alpha^*\,.\,(\lambda\rho'\,.\,\mathcal{C}[\![\Gamma^*]\!]\rho'(\mathcal{E}[\![\mathrm{E}_0]\!]\rho'\kappa')) \\
\quad\quad\quad\quad\quad\quad (\mathit{extends}\,\rho\,\mathrm{I}^*\,\alpha^*)) \\
\quad\quad\quad\quad\quad\quad \epsilon^*, \\
\quad\quad\quad\quad\quad \mathit{wrong}\ \text{“wrong number of arguments”}\rangle \\
\quad\quad\quad\quad \text{in } \mathtt{E}) \\
\quad\quad\quad \kappa \\
\quad\quad\quad (\mathit{update}\,(\mathit{new}\,\sigma \,|\, \mathtt{L})\ \mathit{unspecified}\ \sigma), \\
\quad\quad \mathit{wrong}\ \text{“out of memory”}\ \sigma
\end{array}$$

Action Semantics

- Example: Java (excerpt)

evaluate ⟦ E:Expression "." I:Identifier "(" A:Arguments$^?$ ")" ⟧ =

| evaluate E and then
| respectively evaluate A

then

| | enact the application of
| | the instance-method I of the class of the given object#1
| | to the given (object, value*)
| or
| | check there is given (null-reference, value*) then
| | escape with a throw of . . . .

- the instance methods of $[\![$ M:Modifier* R:("void" | Type) I:Identifier
  "(" F:Formal-Parameters$^?$ ")" T:Throws-Clause$^?$
  B:Block $]\!]$ =

  if "static" is in the set of M then
  the empty-map
  else
  the map of the method-token of I to
  the closure of the abstraction of
  | furthermore
  | | | bind this-token to the given object#1 and
  | | | produce the field-variable-bindings
  | | | of the given object#1
  | | before
  | | | give the rest of the given data then
  | | | respectively formally bind F
  | hence
  | | execute B
  | | trap a return then give the returned-value of it .

A Naïve Action Semantics

- A “lightweight” version
- Facets
- Yielders
- Action with only two combinators
  - ✓ andthen
  - ✓ or

Facets

- a Strachey-like characteristic domain
  - ✓ a collection of data values
- transient facet
  - ✓ short-lived data
  - ✓ produced, copied, and consumed during computation
  - ✓ functional facet (values) + declarative facet (bindings)
- persistent facet
  - ✓ long-lived, fixed data structures
  - ✓ referenced and updated during computation
  - ✓ imperative facet (stores)

Functional Facet

- A monoid of $\mathscr{F}$ = (F, :, < >) where

$$
\begin{aligned}
\mathsf{F} &= \mathsf{List}(\mathsf{Transient}) \\
\mathsf{Transient} &= \mathsf{F} \cup \mathsf{D} \cup \mathsf{Expressible} \cup \mathsf{Closure} \\
\mathsf{Closure} &= \mathsf{Set}(\mathsf{Transient}) \times \mathsf{Identifier} \times \mathsf{Action}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Identifier} &= \text{identifiers} \\
\mathsf{Action} &= \text{text of actions}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Cell} &= \text{storage locations} \\
\mathsf{Int} &= \text{integers}
\end{aligned}
$$

$$
\mathsf{Expressible} = \mathsf{Cell} \cup \mathsf{Int}
$$

  - ✓ : is a sequence append
  - ✓ < > is an empty sequence

- Example

$$
\langle 2, \mathit{cell99}, \{(\mathrm{x}, \mathit{cell99})\} \rangle
$$

Declarative Facet

- A monoid of $\mathcal{D}$ = (D, +, {}) where

$$\mathsf{D} = \mathsf{Set}(\mathsf{Identifier} \times \mathsf{Denotable})$$
$$\mathsf{Denotable} = \mathsf{Transient}$$
$$\mathsf{Identifier} = \mathrm{identifiers}$$

  - ✓ bindings

$$\rho = \{(I_0, n_0), (I_1, n_1), \cdots (I_m, n_m \cdots\}$$

  - ✓ + is a binding override

For values ρ_1 and ρ_2,

$$\rho_1 + \rho_2 = \rho_2 \cup \{(I_j = n_j) \in \rho_1 \mid I_j \notin domain(\rho_2)\}$$

  - ✓ {} is an empty binding

Imperative Facet

- A monoid of $\mathcal{I}$ = (I, *, []) where

$$\sigma = [\ell_0 \mapsto n_0,\ \ell_1 \mapsto n_1,\ \cdots,\ \ell_k \mapsto n_k],$$

$$\ell_i \in \mathsf{Cell} \text{ and } n_i \in \mathsf{Storable}$$

$$\mathsf{I} = \mathsf{Cell} \to \mathsf{Storable} \qquad \mathsf{Cell} = \text{storage locations}$$

$$\mathsf{Storable} = \mathsf{Int} \qquad \mathsf{Int} = \text{integers}$$

$$\mathsf{Expressible} = \mathsf{Cell} \cup \mathsf{Int}$$

Composition, $\sigma_1 * \sigma_2$, is function union

$[\,]$ is the empty map.

Compound Facet

For *distinct* facets, $\mathcal{F} = (F, \circ_F, id_F)$ and $\mathcal{G} = (G, \circ_G, id_G)$,

$$\mathcal{FG} = (\{\{f, g\} \mid f \in F, g \in G\},\ \circ_{FG},\ \{id_F, id_G\})$$
$$\text{where } \{f_1, g_1\} \circ_{FG} \{f_2, g_2\} = \{f_1 \circ_F f_2,\ g_1 \circ_G g_2\}$$

Yielders

- Operations on values within a facet
- Computed at earlier binding times (compile-time)
  - ✓ type checking, constant folding, partial evaluation
- Functional-facet yielders

primitive constant: $\dfrac{}{\vdash \mathsf{k} : k}$

n-ary operation (e.g., addition): $\dfrac{\Gamma \vdash \mathsf{y}_1 : \tau_1 \quad \Delta \vdash \mathsf{y}_2 : \tau_2}{\Gamma \cup \Delta \vdash \mathsf{add}\ \mathsf{y}_1\ \mathsf{y}_2 : add(\tau_1, \tau_2)}$

indexing: $\dfrac{1 \leq i \leq m}{\langle \tau_1, \cdots, \tau_m \rangle \vdash \#\mathsf{i} : \tau_i}$ Note: it abbreviates #1

sort filtering: $\dfrac{\Gamma \vdash \mathsf{y} : \Delta \qquad \Delta \leq T}{\Gamma \vdash \mathsf{is}T\ \mathsf{y} : \Delta}$ where $\leq$ is defined in Section 10

Yielders

- Delcarative-facet yielders
  - ✓ binding lookup $$\frac{(\mathsf{I}, \tau) \in \rho}{\rho \vdash \mathsf{find}\ \mathsf{I} : \tau}$$
  - ✓ binding creation $$\frac{\Gamma \vdash \mathsf{y} : \tau}{\Gamma \vdash \mathsf{bind}\ \mathsf{I}\ \mathsf{y} : \{(\mathsf{I}, \tau)\}}$$
  - ✓ copy $$\frac{}{\rho \vdash \mathsf{currentbindings} : \rho}$$
- Example

$$\frac{\{(\mathrm{x}, n_0), (\mathrm{z}, n_2)\} \vdash \mathsf{find}\ \mathrm{x} : n_0 \qquad \langle n_1 \rangle \vdash \mathsf{it} : n_1}{\{\langle n_1 \rangle, \{(\mathrm{x}, n_0), (\mathrm{z}, n_2)\}\} \vdash \mathsf{add}\ (\mathsf{find}\ \mathrm{x})\ \mathsf{it} : add(n_0, n_1)}$$

Actions

- Compute on facets
- Structural actions

$$\frac{\Gamma \vdash \mathsf{y} : \Delta \qquad \Delta \in \mathcal{G}}{\Gamma \vdash \mathsf{give}_{\mathcal{G}}\ \mathsf{y} \Rightarrow \Delta}$$

$$\frac{}{\vdash \mathsf{complete} \Rightarrow \mathit{completing}}$$

- Imperative-facet actions

$$\frac{c \notin \mathit{domain}(\sigma)}{\sigma \vdash \mathsf{allocate} \Rightarrow \langle c \rangle,\ \sigma * [c \mapsto ?]}$$

$$\frac{\Gamma \vdash \mathsf{y} : c \qquad c \leq \mathsf{Cell}}{\Gamma \cup \sigma \vdash \mathsf{lookup}\ \mathsf{y} \Rightarrow \langle \sigma(c) \rangle}$$

$$\frac{\Gamma_1 \vdash \mathsf{y}_1 : c \qquad c \leq \mathsf{Cell} \qquad \Gamma_2 \vdash \mathsf{y}_2 : \tau \qquad \tau \leq \mathsf{Storable}}{\Gamma_1 \cup \Gamma_2 \cup \sigma \vdash \mathsf{update}\ \mathsf{y}_1\ \mathsf{y}_2 \Rightarrow \sigma[c \mapsto \tau]}$$

Actions

- Closure construction yielder

$$\frac{\Gamma \vdash \mathsf{y} : \Delta}{\Gamma \vdash \mathsf{recabstract}_{\mathcal{G}}\ I\ \mathsf{y}\ \mathsf{a}\ :\ [\Delta{\downarrow_{\mathcal{G}}}, I, \mathsf{a}]_{\mathcal{G}}}$$

where $\mathcal{G}$ names only transient facets

- Closure application action

$$\frac{\begin{array}{l}\Gamma_1 \vdash \mathsf{y}_1 : [\Delta, I, \mathsf{a}]_{\mathcal{G}} \\ \Gamma_2 \vdash \mathsf{y}_2 : \tau \qquad \langle\tau\rangle \circ (\Delta \cup (\Gamma{\downarrow_{\sim\mathcal{G}}})) \circ \{(I, [\Delta, I, \mathsf{a}]_{\mathcal{G}})\} \vdash \mathsf{a} \Rightarrow \Sigma \\ \Gamma = \Gamma_1 \cup \Gamma_2\end{array}}{\Gamma \vdash \mathsf{exec}\ \mathsf{y}_1\ \mathsf{y}_2 \Rightarrow \Sigma}$$

Note: y_2 is optional.

- Example

$$\mathsf{recabstract}_{\mathcal{D}}\ \mathsf{f}\ \mathsf{currentbindings}\ (\mathsf{give}_{\mathcal{F}}(\mathsf{add}\,(\mathsf{find}\,x)\ \mathsf{it}))$$

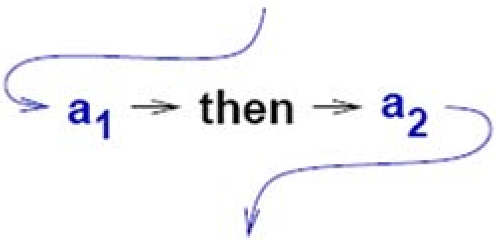
Weakening & Strengthening Rules

- weaken-L: $$\frac{\Gamma \vdash \mathsf{a} \Rightarrow \Delta}{\Sigma \cup \Gamma \vdash \mathsf{a} \Rightarrow \Delta}$$
  - ✓ ac action can consume more facets than what are needed to construct an action, no harm occurs
- strengthen-R: $$\frac{\Gamma \vdash \mathsf{a} \Rightarrow \Delta \qquad \Gamma \cup \sigma = \Gamma}{\Gamma \vdash \mathsf{a} \Rightarrow \Delta \cup \sigma} \quad \text{where } \sigma \in \mathcal{I}$$
  - ✓ an action whose input includes a persistent value, passes forwards that value unaltered.
- Example: $\{(\mathsf{x}, 2)\} \vdash \mathsf{give_F}(\mathsf{find}\ \mathsf{x}) \Rightarrow 2$
  - ✓ weaken-L $\langle\,\rangle, \{(\mathsf{x}, 2)\}, \sigma_0 \vdash \mathsf{give_F}(\mathsf{find}\ \mathsf{x}) \Rightarrow 2$
  - ✓ strengthen-R $\langle\,\rangle, \{(\mathsf{x}, 2)\}, \sigma_0 \vdash \mathsf{give_F}(\mathsf{find}\ \mathsf{x}) \Rightarrow 2, \sigma_0$

Facet Flows

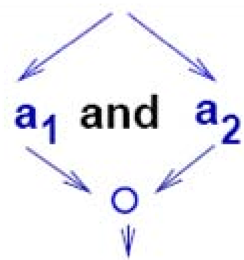
- Sequential

$$\frac{\Gamma \vdash \mathsf{a}_1 \Rightarrow \Delta \quad \Delta \vdash \mathsf{a}_2 \Rightarrow \Sigma}{\Gamma \vdash \mathsf{a}_1 \text{ then } \mathsf{a}_2 \Rightarrow \Sigma}$$



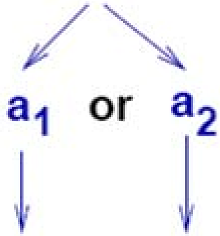
- Parallel

$$\frac{\Gamma \vdash \mathsf{a}_1 \Rightarrow \Delta_1 \quad \Gamma \vdash \mathsf{a}_2 \Rightarrow \Delta_2}{\Gamma \vdash \mathsf{a}_1 \text{ and } \mathsf{a}_2 \Rightarrow \Delta_1 \circ \Delta_2}$$



- Conditional

$$\frac{\Gamma \vdash \mathsf{a}_i \Rightarrow \Delta \quad i \in \{1, 2\}}{\Gamma \vdash \mathsf{a}_1 \text{ or } \mathsf{a}_2 \Rightarrow \Delta}$$



Universal Combinator

- and$_{\mathcal{G}}$then
  - ✓ $\mathcal{G}$ denotes the (compound) facet that is passed in parallel
  - ✓ all other facets are passed sequentially

$$\frac{\Gamma \vdash \mathsf{a}_1 \Rightarrow \Delta_1 \qquad (\Gamma \downarrow_{\mathcal{G}}) \cup (\Delta_1 \downarrow_{\sim\mathcal{G}}) \vdash \mathsf{a}_2 \Rightarrow \Delta_2}{\Gamma \vdash \mathsf{a}_1\ \mathsf{and}_{\mathcal{G}}\mathsf{then}\ \mathsf{a}_2 \Rightarrow \Delta_1 \downarrow_{\mathcal{G}} \circ \Delta_2}$$

- Full vs. Naïve
  - ✓ then = $\mathsf{and}_{\emptyset}\mathsf{then}$
  - ✓ and = $\mathsf{and}_{\mathsf{AllFacets}}\mathsf{then}$

Example Language Syntax

Expression: E ::= $\mathtt{k} \mid E_1 + E_2 \mid N$

Command: C ::= $N \mathtt{:=} E \mid C_1;C_2 \mid \mathtt{while}\ E\ \mathtt{do}\ C \mid D\ \mathtt{in}\ C \mid \mathtt{call}\ N(E)$

Declaration: D ::= $\mathtt{val}\ I = E \mid \mathtt{var}\ I = E \mid \mathtt{proc}\ I_1(I_2) = C \mid \mathtt{module}\ I = D \mid D_1;D_2$

Name: N ::= $I \mid N.I$

Identifier: I

Action Equations for Expression

$$\text{evaluate} : \text{Expression} \rightarrow \mathcal{DI} \rightarrow \mathcal{F}$$

$$\text{evaluate}[\![\texttt{k}]\!] = \text{give}_{\mathcal{F}}\ \text{k}$$

$$\text{evaluate}[\![E_1 + E_2]\!] = \begin{array}{l}(\text{evaluate } E_1 \text{ and}_{\mathcal{FD}}\text{then evaluate } E_2) \\ \text{andthen give}_{\mathcal{F}}(\text{add (isInt \#1) (isInt \#2))}\end{array}$$

$$\text{evaluate}[\![N]\!] = \text{investigate } N \text{ andthen } \begin{array}{l}\text{lookup (isCell it)} \\ \text{or give}_{\mathcal{F}} \text{ (isInt it)}\end{array}$$

$$\text{investigate} : \text{Name} \rightarrow \mathcal{D} \rightarrow \mathcal{F}$$

$$\text{investigate}[\![I]\!] = \text{give}_{\mathcal{F}}(\text{find } I)$$

$$\text{investigate}[\![N.I]\!] = \text{investigate } N \text{ then give}_{\mathcal{D}}(\text{isD it}) \text{ then give}_{\mathcal{F}}(\text{find } I)$$

Action Equations for Command

$$\text{execute} : \text{Command} \rightarrow \mathcal{DI} \rightarrow \mathcal{I}$$

$$\text{execute}[\![N := E]\!] = \begin{array}{l}(\text{investigate } N \text{ and}_{\mathcal{FD}}\text{then evaluate } E) \\ \text{ andthen update(isCell \#1) \#2}\end{array}$$

$$\text{execute}[\![C_1 ; C_2]\!] = \text{execute } C_1 \text{ andthen execute } C_2$$

$$\text{execute}[\![\texttt{while } E \text{ do } C]\!] = \begin{array}{l}\text{evaluate } E \text{ andthen} \\ \text{ ((give}_{\mathcal{F}}\text{(isZero it) andthen complete)} \\ \text{ or} \\ \text{ (give}_{\mathcal{F}}\text{(isNonZero it) andthen execute } C \\ \text{ andthen execute } [\![\texttt{while } E \text{ do } C]\!]))\end{array}$$

$$\text{execute}[\![D \texttt{ in } C]\!] = (\text{give}_{\mathcal{D}} \text{ currentbindings andthen elaborate } D) \text{ then execute } C$$

$$\text{execute}[\![\texttt{call } N(E)]\!] = \begin{array}{l}(\text{investigate } N \text{ and}_{\mathcal{FD}}\text{then evaluate } E) \\ \text{ andthen exec(isClosure \#1) \#2}\end{array}$$

Action Equations for Declaration

$$\text{elaborate} : \text{Declaration} \rightarrow \mathcal{DI} \rightarrow \mathcal{DI}$$

$$\text{elaborate}[\![\texttt{val}\ I = E]\!] = \text{evaluate } E \text{ andthen give}_{\mathcal{D}} \text{ (bind } I \text{ it)}$$

$$\text{elaborate}[\![\texttt{var}\ I = E]\!] = \begin{array}{l}\text{(evaluate } E \text{ and}_{\mathcal{FD}}\text{then allocate)} \\ \ \text{andthen (give}_{\mathcal{D}} \text{ (bind } I\ \#2) \text{ and}_{\mathcal{FD}}\text{then update } \#2\ \#1)\end{array}$$

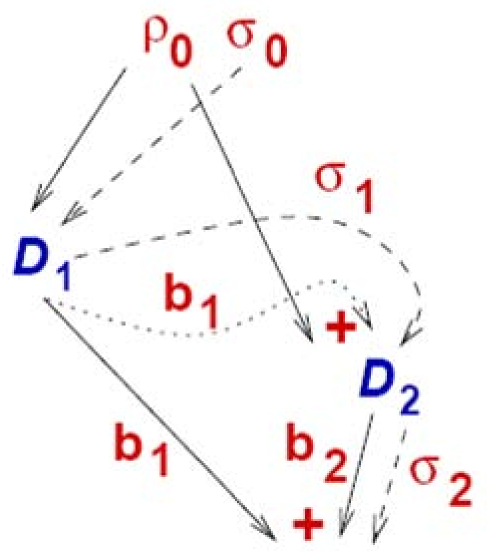
$$\text{elaborate}[\![\texttt{proc}\ I_1(I_2) = C]\!] = \text{give}_{\mathcal{D}}(\text{bind } I_1\ \mathit{closure})$$

$$\text{where } \mathit{closure} = \text{recabstract}_{\mathcal{D}}\ I_1 \text{ (currentbindings)} \begin{array}{l}\text{((give}_{\mathcal{D}} \text{ currentbindings} \\ \ \text{and}_{\mathcal{FD}}\text{then give}_{\mathcal{D}}(\text{bind } I_2 \text{ it))} \\ \text{then execute } C)\end{array}$$

$$\text{elaborate}[\![\texttt{module}\ I = D]\!] = \text{elaborate } D \text{ then give}_{\mathcal{D}}(\text{bind } I \text{ currentbindings})$$

$$\text{elaborate}[\![D_1 ; D_2]\!] = \begin{array}{l}\text{(elaborate } D_1 \\ \qquad \text{then (give}_{\mathcal{F}} \text{ currentbindings and give}_{\mathcal{D}} \text{ currentbindings))} \\ \text{andthen ((give}_{\mathcal{D}} \text{ currentbindings and}_{\mathcal{FD}}\text{then give}_{\mathcal{D}} \text{ it)} \\ \qquad\qquad \text{then elaborate } D_2)\end{array}$$

Facet Flow of $D_1; D_2$



$$\mathcal{D}[\![D_1; D_2]\!]\rho_0\sigma_0 = \mathit{let}\ b_1, \sigma_1 = \mathcal{D}[\![D_1]\!]\rho_0\sigma_0 \\ \mathit{let}\ b_2, \sigma_2 = \mathcal{D}[\![D_2]\!](\rho_0 + b_1)\sigma_1 \\ \mathit{in}\ b_1 + b_2, \sigma_2$$

$\text{elaborate}[\![D_1; D_2]\!] =$

elaborate D_1 before elaborate D_2

$\text{elaborate}[\![D_1; D_2]\!] =$

(elaborate D_1
then (give$_\mathcal{F}$ currentbindings and give$_\mathcal{D}$ currentbindings))
andthen ((give$_\mathcal{D}$ currentbindings and$_{\mathcal{F}\mathcal{D}}$then give$_\mathcal{D}$ it)
then elaborate D_2)

Mosses Abstraction

$$(\mathsf{p} : \mathcal{G}) \Rightarrow \mathsf{a}$$

- Example

$$\mathsf{give}_{\mathcal{F}}(\mathsf{add}\ \#2\ (\mathsf{find}\ \mathrm{x}))$$

$$\Downarrow$$

$$(\langle \mathsf{v}, \mathsf{w} \rangle : \mathcal{F}) \Rightarrow (\{(\mathrm{x}, \mathsf{d})\} : \mathcal{D}) \Rightarrow \mathsf{give}_{\mathcal{F}}(\mathsf{add}\ \mathsf{w}\ \mathsf{d})$$

Mosses Abstraction

$$\text{elaborate}[\![\texttt{proc}\ I_1(I_2) = C]\!] = \text{give}_{\mathcal{D}}(\text{bind}\ I_1\ \mathit{closure})$$

$$\text{where}\ \mathit{closure} = \text{recabstract}_{\mathcal{D}}\ I_1\ (\text{currentbindings})\ \begin{array}{l}((\text{give}_{\mathcal{D}}\ \text{currentbindings} \\ \ \ \text{and}_{\mathcal{FD}}\text{then give}_{\mathcal{D}}(\text{bind}\ I_2\ \text{it})) \\ \ \text{then execute}\ C)\end{array}$$

⇓

$$\text{elaborate}[\![\texttt{proc}\ I_1(I_2) = C]\!] = (\text{rho} : \mathcal{D})\ \texttt{=>}\ \text{give}_{\mathcal{D}}(\text{bind}\ I_1\ \mathit{closure})$$

$$\begin{array}{l}\text{where}\ \mathit{closure} = \text{recabstract}_{\mathcal{D}}\ I_1\ \text{rho} \\ \qquad (((\langle\text{arg}\rangle : \mathcal{F})\ \texttt{=>} \\ \qquad (\text{give}_{\mathcal{D}}\ \text{rho}\ \text{and}_{\mathcal{FD}}\text{then give}_{\mathcal{D}}(\text{bind}\ I_2\ \text{arg})) \\ \qquad \text{then execute}\ C)\end{array}$$

Mosses Abstraction

$$\text{elaborate}[\![D_1; D_2]\!] =$$

$$
\begin{aligned}
&(\text{elaborate } D_1 \\
&\quad \text{then } (\text{give}_{\mathcal{F}} \text{ currentbindings and give}_{\mathcal{D}} \text{ currentbindings})) \\
&\text{andthen } ((\text{give}_{\mathcal{D}} \text{ currentbindings and}_{\mathcal{F}\mathcal{D}}\text{then give}_{\mathcal{D}} \text{ it}) \\
&\qquad\quad \text{then elaborate } D_2)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\text{elaborate}[\![D_1; D_2]\!] = (\text{rho}_0 : \mathcal{D}) \text{ => } \\
&\quad (\text{elaborate } D_1 \text{ then } (\text{rho}_1 : \mathcal{D}) \text{ => give}_{\mathcal{F}} \text{ rho}_1 \text{ and give}_{\mathcal{D}} \text{ rho}_1) \\
&\quad \text{andthen} \\
&\quad ((\langle \text{rho}_1 \rangle : \mathcal{F}) \text{ => } (\text{give}_{\mathcal{D}} \text{ rho}_0 \text{ and}_{\mathcal{F}\mathcal{D}}\text{then give}_{\mathcal{D}} \text{ rho}_1) \text{ then elaborate } D_2)
\end{aligned}
$$

Things to do

- Implementation in Haskell
- Theory of action equivalence
- Specify the semantics of real-life programming languages in action semantics

- Anyone?