Call-by-name vs. call-by-value lambda calculus

Normal β-reduction rule implements **call-by-name** parameter passing

- pass in argument expressions to be evaluated when needed

β-rule: $(\lambda I.E_1)E_2 \Leftrightarrow [E_2/I]E_1$

Can define alternative β_{value}-reduction rule implementing **call-by-value** parameter passing

- force evaluation of arguments before passing in
- β_{value}-redexes $\subseteq$ β-redexes

β_{value}-rule: $(\lambda I.E_1)E_2 \Leftrightarrow [E_2/I]E_1$ **if E_2 is a value**

I and $(\lambda I.E)$ are values, but not $E_1\ E_2$

Kinds of semantics

Static semantics: the static properties of the program

- essentially, typechecking rules

Dynamic semantics: the dynamic behavior of the program

- when you run the program, what result do you get?
- often defined only for programs for which the static semantics is defined (i.e. those programs that typecheck)

Flavors of dynamic semantics:

- operational semantics
  - specified as evaluation steps as by a machine
  - simple, easy to build an interpreter from
- denotational semantics
  - specified directly in terms of mathematical functions
  - more abstract than operational semantics
- axiomatic semantics
  - specified via pre- and post-conditions
  - good for doing behavioral specification & correctness proofs

Operational semantics

Previously defined execution of a λ-calculus expression by rewriting, using β-reduction

This is a kind of **small step** operational semantics

- each inference step represents one atomic step of the underlying machine

Can also express using natural deduction notation

Evaluation judgments of the form: $E_1 \Rightarrow E_2$

- no context needed, for this simple language

[⇒] $(\lambda I{:}E_1)E_2 \Rightarrow [E_2/I]E_1$

[λ] $\dfrac{E \Rightarrow E'}{\lambda I{:}E \Rightarrow \lambda I{:}E'}$

[app_1] $\dfrac{E_1 \Rightarrow E_1'}{E_1\ E_2 \Rightarrow E_1'\ E_2}$

[app_2] $\dfrac{E_2 \Rightarrow E_2'}{E_1\ E_2 \Rightarrow E_1\ E_2'}$

Call-by-value version

[⇒] $(\lambda I{:}E_1)E_2 \Rightarrow [E_2/I]E_1$ if E_2 is a value

[app_1] $\dfrac{E_1 \Rightarrow E_1'}{E_1\ E_2 \Rightarrow E_1'\ E_2}$ if E_2 is a value

[app_2] $\dfrac{E_2 \Rightarrow E_2'}{E_1\ E_2 \Rightarrow E_1\ E_2'}$

Do we want this one, too?

[λ] $\dfrac{E \Rightarrow E'}{\lambda I{:}E \Rightarrow \lambda I{:}E'}$

Natural semantics

Can be tedious to write semantics for individual steps
(e.g. app_1 vs. app_2 rules)

Alternative: big-step operational semantics,
a.k.a. **natural semantics**

Each evaluation judgment will be that some kind of expression E evaluates *completely* to some result value V, not just in a single step to a slightly simpler E′

Most appropriate for call-by-value languages, where the notion of fully evaluating a subexpression makes most sense

Evaluation judgments: $E \Rightarrow V$
meaning E evaluates fully to value (normal form) V

[var] $\quad I \Rightarrow I$

[λ] $\quad (\lambda I.E) \Rightarrow (\lambda I.E)$

[app] $$\frac{E_1 \Rightarrow \lambda I{:}E \qquad E_2 \Rightarrow V_2 \qquad [V_2/I]E \Rightarrow V}{(E_1\ E_2) \Rightarrow V}$$

Introducing environments

Substitution is not so nice
- somewhat complicated semantically
- don't want to implement evaluation by rewriting program

Can reexpress by adding an **environment** ρ as context, parallel to typing environment
- environment maps identifiers to their values
- to get scoping right, values need to be closures

Evaluation judgments: $\rho \vdash E \Rightarrow V$ if
in environment ρ the expression E evaluates to value V

[var] $\quad \rho \vdash I \Rightarrow V \qquad \text{if } I{:}V \in \rho$

[λ] $\quad \rho \vdash (\lambda I.E) \Rightarrow [(\lambda I.E), \rho]$

[app] $$\frac{\rho \vdash E_1 \Rightarrow [(\lambda I.E), \rho'] \qquad \rho \vdash E_2 \Rightarrow V_2 \qquad \rho' \cup I{:}V_2 \vdash E \Rightarrow V}{\rho \vdash (E_1\ E_2) \Rightarrow V}$$

This is normal way to express operational semantics for a call-by-value functional language

Subject reduction

We want static and dynamic semantics to be consistent

For an operational semantics w.r.t. a static semantics:
subject reduction theorem
- reduction/evaluation preserves type

For small-step operational semantics:
if $\Gamma \vdash E : \tau$ and $E \Rightarrow^* E'$, then $\Gamma \vdash E' : \tau$

For natural semantics:
if $\Gamma \vdash E : \tau$ and $\rho \vdash E \Rightarrow V$, and ρ is compatible with Γ w.r.t. E,
then $\Gamma \vdash V : \tau$

Denotational semantics

Denotational semantics associates each language construct with a mathematical function giving its meaning
- compositional: meaning of a construct defined by composing meaning of subexpressions, not based on earlier execution

Operational semantics ≈ an interpreter

Denotational semantics ≈ a compiler (into math)

Untyped semantics

$\mathcal{E}$ is the function which gives a meaning to each construct, as a function from an environment to a mathematical value

- define by cases in syntax-directed way

For untyped lambda calculus:

$\text{Env} \equiv \text{Id} \rightarrow \text{Val}$

$\text{Val} \equiv \text{Val} \rightarrow \text{Val}$ (!!!)

$\mathcal{E}$: $\mathtt{E} \rightarrow \text{Env} \rightarrow \text{Val}$

$\mathcal{E}[\![\mathtt{I}]\!]\ \rho \equiv \rho(\mathtt{I})$

$\mathcal{E}[\![\lambda\mathtt{I.E}]\!]\ \rho \equiv \lambda\mathtt{E'}.\ \mathcal{E}[\![\mathtt{E}]\!]\ (\rho \cup \{\mathtt{I{:}E'}\})$

$\mathcal{E}[\![\mathtt{E}_1\ \ \mathtt{E}_2]\!]\ \rho \equiv (\mathcal{E}[\![\mathtt{E}_1]\!]\ \rho)(\mathcal{E}[\![\mathtt{E}_2]\!]\ \rho)$

Compositional:
each $\mathcal{E}$ case is defined in terms of $\mathcal{E}$ of subexpressions

$\mathcal{E}$ is recursively defined....

Val is weird....

Typed semantics

For simply typed lambda calculus,
give meaning only to *typed* expressions, and
define separate evaluation for each type of Val

$\text{Env} \equiv \text{Id}_\tau \rightarrow \text{Val}_\tau$

$\mathcal{E}_\tau$: $\mathtt{E}_\tau \rightarrow \text{Env} \rightarrow \text{Val}_\tau$

$\mathcal{E}_\tau [\![\mathtt{I}_\tau]\!]\ \rho \equiv \rho(\mathtt{I}_\tau)$

$\mathcal{E}_{\tau 1\rightarrow\tau 2} [\![(\lambda\mathtt{I}:\tau_1.\mathtt{E}_{\tau 2})_{\tau 1\rightarrow\tau 2}]\!]\ \rho \equiv \lambda\mathtt{E'}.\ \mathcal{E}_{\tau 2} [\![\mathtt{E}_{\tau 2}]\!]\ (\rho \cup \{\mathtt{I}_{\tau 1}\rightarrow\mathtt{E'}\})$

$\mathcal{E}_{\tau 2} [\![(\mathtt{E}_{\tau 1\rightarrow\tau 2}\ \ \mathtt{E'}_{\tau 1})_{\tau 2}]\!]\ \rho \equiv (\mathcal{E}_{\tau 1\rightarrow\tau 2} [\![\mathtt{E}_{\tau 1\rightarrow\tau 2}]\!]\ \rho)(\mathcal{E}_{\tau 1} [\![\mathtt{E'}_{\tau 1}]\!]\ \rho)$

Val_τ stands for the set of all values of type τ:

$\text{Val}_* \equiv \{*\}$

$\text{Val}_{\tau 1\rightarrow\tau 2} \equiv \text{Val}_{\tau 1} \rightarrow \text{Val}_{\tau 2}$

Now each $\mathcal{E}$ rule is defined in terms of simpler $\mathcal{E}$ rules, and
the Val domains are defined in terms of simpler Val domains

- $\mathcal{E}$ rules can be macro-expanded to give a direct definition w/o calls for each construct with a given type
- nice set-theoretic interpretation of Val domains

Soundness of typing

We want static semantics, operational semantics, and
denotational semantics to be consistent:

For a denotational semantics w.r.t. a static semantics:
soundness theorem

- meaning-attribution preserves type

if $\Gamma \vdash \mathtt{E} : \tau$ and $\mathcal{E}_\tau [\![\mathtt{E}_\tau]\!]\ \rho \equiv v$, and ρ is compatible with Γ w.r.t. $\mathtt{E}$,
then $v \in \text{Val}_\tau$

Soundness of evaluation

For a denotational semantics w.r.t. an operational semantics:
another kind of **soundness** or **agreement** theorem

- evaluation preserves meaning

Applicable even in untyped lambda calculus

For small-step operational semantics:
if $\mathcal{E}[\![\mathtt{E}]\!]\ \rho \equiv v$ and $\mathtt{E} \Rightarrow^* \mathtt{E'}$, then $\mathcal{E}[\![\mathtt{E'}]\!]\ \rho \equiv v$

For natural semantics:
if $\mathcal{E}[\![\mathtt{E}]\!]\ \rho \equiv v$ and $\rho \vdash \mathtt{E} \Rightarrow \mathtt{V}$, then $\mathcal{E}[\![\mathtt{V}]\!]\ \rho \equiv v$

A fuller language: PCF

Simply typed lambda calculus plus:

- a fixpoint operator (`rec`) to get needed expressiveness within simply typed framework
- a name binding form (`let`) for convenience
- some additional basic values (numbers, booleans), corresponding types (`nat`, `bool`), and operations (+, =, `not`, `if`)
- some additional value constructors (pairs, unions), corresponding type constructors (×, +), and operations

```
E    ::= I | λI:τ.E | E E
       | rec I:τ = E
       | let I = E in E
       | N | E+E | E=E | ...
       | true | false | not E | ...
       | if E then E else E
       | pair E E | first E | second E
       | mkfirst_τ+τ E | mksecond_τ+τ E
       | isfirst E | issecond E
       | getfirst E | getsecond E
τ    ::= nat | bool | τ→τ | τ×τ | τ+τ
```


Typing rules

Identifiers:

[var] $\Gamma \vdash \mathtt{I}:\tau \quad \text{if } \mathtt{I}:\tau \in \Gamma$

Functions:

[→ intro] $$\frac{\Gamma \cup \mathtt{I}:\tau_1 \vdash \mathtt{E}:\tau_2}{\Gamma \vdash (\lambda \mathtt{I}:\tau_1.\mathtt{E}):\tau_1 \to \tau_2}$$

[→ elim] $$\frac{\Gamma \vdash \mathtt{E}_1:\tau_1 \to \tau_2 \qquad \Gamma \vdash \mathtt{E}_2:\tau_1}{\Gamma \vdash (\mathtt{E}_1\ \mathtt{E}_2):\tau_2}$$

Recursion:

[rec] $$\frac{\Gamma \cup \mathtt{I}:\tau \vdash \mathtt{E}:\tau}{\Gamma \vdash (\mathtt{rec\ I}:\tau\ \mathtt{=\ E}):\tau}$$

Name binding:

[let] $$\frac{\Gamma \vdash \mathtt{E}_1:\tau_1 \qquad \Gamma \cup \mathtt{I}:\tau_1 \vdash \mathtt{E}_2:\tau_2}{\Gamma \vdash (\mathtt{let\ I\ =\ E}_1\ \mathtt{in\ E}_2):\tau_2}$$

Numbers:

[num] $\Gamma \vdash \mathtt{N}:\mathtt{nat}$

[+] $$\frac{\Gamma \vdash \mathtt{E}_1:\mathtt{nat} \qquad \Gamma \vdash \mathtt{E}_2:\mathtt{nat}}{\Gamma \vdash (\mathtt{E}_1+\mathtt{E}_2):\mathtt{nat}}$$

[=] $$\frac{\Gamma \vdash \mathtt{E}_1:\mathtt{nat} \qquad \Gamma \vdash \mathtt{E}_2:\mathtt{nat}}{\Gamma \vdash (\mathtt{E}_1=\mathtt{E}_2):\mathtt{bool}}$$

Booleans:

[true] $\Gamma \vdash \mathtt{true}:\mathtt{bool}$

[false] $\Gamma \vdash \mathtt{false}:\mathtt{bool}$

[not] $$\frac{\Gamma \vdash \mathtt{E}:\mathtt{bool}}{\Gamma \vdash (\mathtt{not\ E}):\mathtt{bool}}$$

[if] $$\frac{\Gamma \vdash \mathtt{E}_1:\mathtt{bool} \qquad \Gamma \vdash \mathtt{E}_2:\tau \qquad \Gamma \vdash \mathtt{E}_3:\tau}{\Gamma \vdash (\mathtt{if\ E}_1\ \mathtt{then\ E}_2\ \mathtt{else\ E}_3):\tau}$$

Pairs:

[× intro] $$\frac{\Gamma \vdash \mathtt{E}_1:\tau_1 \qquad \Gamma \vdash \mathtt{E}_2:\tau_2}{\Gamma \vdash (\mathtt{pair\ E}_1\ \mathtt{E}_2):\tau_1 \times \tau_2}$$

[× elim_1] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1 \times \tau_2}{\Gamma \vdash (\mathtt{first\ E}):\tau_1}$$

[× elim_2] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1 \times \tau_2}{\Gamma \vdash (\mathtt{second\ E}):\tau_2}$$

Unions:

[+ intro_1] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1}{\Gamma \vdash \left(\mathtt{mkfirst}_{\tau_1+\tau_2}\ \mathtt{E}\right):\tau_1+\tau_2}$$

[+ intro_2] $$\frac{\Gamma \vdash \mathtt{E}:\tau_2}{\Gamma \vdash \left(\mathtt{mksecond}_{\tau_1+\tau_2}\ \mathtt{E}\right):\tau_1+\tau_2}$$

[+ pred_1] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1+\tau_2}{\Gamma \vdash (\mathtt{isfirst\ E}):\mathtt{bool}}$$

[+ pred_2] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1+\tau_2}{\Gamma \vdash (\mathtt{issecond\ E}):\mathtt{bool}}$$

[+ elim_1] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1+\tau_2}{\Gamma \vdash (\mathtt{getfirst\ E}):\tau_1}$$

[+ elim_2] $$\frac{\Gamma \vdash \mathtt{E}:\tau_1+\tau_2}{\Gamma \vdash (\mathtt{getsecond\ E}):\tau_2}$$

Natural operational semantics

Identifiers:

[var] $\quad \rho \vdash \mathtt{I} \rightarrow \mathtt{V} \qquad \text{if } \mathtt{I}{:}\mathtt{V} \in \rho$

Functions:

[λ] $\quad \rho \vdash (\lambda \mathtt{I}.\mathtt{E}) \rightarrow [(\lambda \mathtt{I}.\mathtt{E}), \rho]$

$$[\mathrm{app}] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow [(\lambda \mathtt{I}.\mathtt{E}), \rho'] \qquad \rho \vdash \mathtt{E}_2 \rightarrow \mathtt{V}_2 \qquad \rho' \cup \mathtt{I}{:}\mathtt{V}_2 \vdash \mathtt{E} \rightarrow \mathtt{V}}{\rho \vdash (\mathtt{E}_1\ \mathtt{E}_2) \rightarrow \mathtt{V}}$$

Recursion:

$$[\mathrm{rec}] \quad \frac{\rho \cup \mathtt{I}{:}\mathtt{V} \vdash \mathtt{E} \rightarrow \mathtt{V}}{\rho \vdash (\mathtt{rec}\ \mathtt{I}{:}\tau\ \mathtt{=}\ \mathtt{E}) \rightarrow \mathtt{V}}$$

- hmmm....
- small-step semantics: `rec I:τ=E` $\Rightarrow$ `[rec I:τ=E/I]E`
  - the **unfolding** rule

Name binding:

$$[\mathrm{let}] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow \mathtt{V}_1 \qquad \rho \cup \mathtt{I}{:}\mathtt{V}_1 \vdash \mathtt{E}_2 \rightarrow \mathtt{V}_2}{\rho \vdash (\mathtt{let}\ \mathtt{I}\ \mathtt{=}\ \mathtt{E}_1\ \mathtt{in}\ \mathtt{E}_2) \rightarrow \mathtt{V}_2}$$

Numbers:

[num] $\quad \rho \vdash \mathtt{N} \rightarrow \mathrm{N} \qquad$ where N is semantic equivalent of `N`

$$[+] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow \mathtt{V}_1 \qquad \rho \vdash \mathtt{E}_2 \rightarrow \mathtt{V}_2}{\rho \vdash (\mathtt{E}_1 \mathtt{+} \mathtt{E}_2) \rightarrow (\mathtt{V}_1 + \mathtt{V}_2)}$$

$$[=] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow \mathtt{V}_1 \qquad \rho \vdash \mathtt{E}_2 \rightarrow \mathtt{V}_2}{\rho \vdash (\mathtt{E}_1 \mathtt{=} \mathtt{E}_2) \rightarrow (\mathtt{V}_1 = \mathtt{V}_2)}$$

- difference between *syntactic* `N`, `+`, `=` and *semantic* N, +, =

Booleans:

[true] $\quad \rho \vdash \mathtt{true} \rightarrow \mathrm{true}$

[false] $\quad \rho \vdash \mathtt{false} \rightarrow \mathrm{false}$

$$[\mathrm{not}] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \mathtt{V}}{\rho \vdash (\mathtt{not}\ \mathtt{E}) \rightarrow (\mathrm{not}\ \mathtt{V})}$$

$$[\mathrm{if}_{\mathrm{true}}] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow \mathrm{true} \qquad \rho \vdash \mathtt{E}_2 \rightarrow \mathtt{V}}{\rho \vdash (\mathtt{if}\ \mathtt{E}_1\ \mathtt{then}\ \mathtt{E}_2\ \mathtt{else}\ \mathtt{E}_3) \rightarrow \mathtt{V}}$$

$$[\mathrm{if}_{\mathrm{false}}] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow \mathrm{false} \qquad \rho \vdash \mathtt{E}_3 \rightarrow \mathtt{V}}{\rho \vdash (\mathtt{if}\ \mathtt{E}_1\ \mathtt{then}\ \mathtt{E}_2\ \mathtt{else}\ \mathtt{E}_3) \rightarrow \mathtt{V}}$$

- difference between *syntactic* `true`, `false`, `not` and *semantic* true, false, not

Pairs:

$$[\times\ \mathrm{intro}] \quad \frac{\rho \vdash \mathtt{E}_1 \rightarrow \mathtt{V}_1 \qquad \rho \vdash \mathtt{E}_2 \rightarrow \mathtt{V}_2}{\rho \vdash (\mathtt{pair}\ \mathtt{E}_1\ \mathtt{E}_2) \rightarrow (\mathtt{V}_1, \mathtt{V}_2)}$$

$$[\times\ \mathrm{elim}_1] \quad \frac{\rho \vdash \mathtt{E} \rightarrow (\mathtt{V}_1, \mathtt{V}_2)}{\rho \vdash (\mathtt{first}\ \mathtt{E}) \rightarrow \mathtt{V}_1}$$

$$[\times\ \mathrm{elim}_2] \quad \frac{\rho \vdash \mathtt{E} \rightarrow (\mathtt{V}_1, \mathtt{V}_2)}{\rho \vdash (\mathtt{second}\ \mathtt{E}) \rightarrow \mathtt{V}_2}$$

Unions:

$$[+\ \mathrm{intro}_1] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \mathtt{V}}{\rho \vdash \left(\mathtt{mkfirst}_{\tau_1 + \tau_2}\ \mathtt{E}\right) \rightarrow \langle \mathrm{first}{=}\mathtt{V} \rangle}$$

$$[+\ \mathrm{intro}_2] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \mathtt{V}}{\rho \vdash \left(\mathtt{mksecond}_{\tau_1 + \tau_2}\ \mathtt{E}\right) \rightarrow \langle \mathrm{second}{=}\mathtt{V} \rangle}$$

$$\left[+\ \mathrm{pred}_{1_{\mathrm{true}}}\right] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \langle \mathrm{first}{=}\mathtt{V} \rangle}{\rho \vdash (\mathtt{isfirst}\ \mathtt{E}) \rightarrow \mathrm{true}}$$

$$\left[+\ \mathrm{pred}_{1_{\mathrm{false}}}\right] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \langle \mathrm{second}{=}\mathtt{V} \rangle}{\rho \vdash (\mathtt{isfirst}\ \mathtt{E}) \rightarrow \mathrm{false}}$$

$$\left[+\ \mathrm{pred}_{2_{\mathrm{false}}}\right] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \langle \mathrm{first}{=}\mathtt{V} \rangle}{\rho \vdash (\mathtt{issecond}\ \mathtt{E}) \rightarrow \mathrm{false}}$$

$$\left[+\ \mathrm{pred}_{2_{\mathrm{true}}}\right] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \langle \mathrm{second}{=}\mathtt{V} \rangle}{\rho \vdash (\mathtt{issecond}\ \mathtt{E}) \rightarrow \mathrm{true}}$$

$$[+\ \mathrm{elim}_1] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \langle \mathrm{first}{=}\mathtt{V} \rangle}{\rho \vdash (\mathtt{getfirst}\ \mathtt{E}) \rightarrow \mathtt{V}}$$

$$[+\ \mathrm{elim}_2] \quad \frac{\rho \vdash \mathtt{E} \rightarrow \langle \mathrm{second}{=}\mathtt{V} \rangle}{\rho \vdash (\mathtt{getsecond}\ \mathtt{E}) \rightarrow \mathtt{V}}$$

- error cases aren't defined $\Rightarrow$ evaluation gets "stuck"