

Environments

Johannes Åman Pohjola
UNSW
Term 3 2022

Where we're at

- We refined the abstract M-Machine to a **C-Machine**, with explicit stacks:

$$s \succ e \quad s \prec v$$

- Function application is still executed via substitution:

$$\overline{(\text{Apply } \langle\langle f.x. e \rangle\rangle \square) \triangleright s \prec v \mapsto_C s \succ e[x := v, f := (\text{Fun } (f.x.e))]}$$

- We're going to extend our C-Machine to replace substitutions with an **environment**, giving us a new **E-Machine**

Environments

Definition

An *environment* is a *context* containing the values of variables.

It is like the *states* of TinyImp, except the value of a variable never changes.

$$\frac{}{\bullet \text{ Env}} \quad \frac{\eta \text{ Env}}{x = v, \eta \text{ Env}}$$

$\eta(x)$ denotes the *leftmost* value bound to x in η .

Let's change our machine states to include an *environment*:

$$s \mid \eta \succ e \quad s \mid \eta \prec v$$

First Attempt

First, we'll add a rule for consulting the environment if we encounter a free variable:

$$\frac{}{s \mid \eta \succ x \mapsto_E s \mid \eta \prec \eta(x)}$$

Then, we just need to handle function application.

One broken attempt:

$$\frac{}{(\text{Apply } \langle\langle f.x. e \rangle\rangle \square) \triangleright s \mid \eta \prec v \mapsto_E s \mid (x = v, f = \langle\langle f.x. e \rangle\rangle, \eta) \succ e}$$

We don't know when to remove the variables again!

Second Attempt

We will extend our **stacks** to allow us to **save** the old environment to it.

$$\frac{\eta \text{ Env } s \text{ Stack}}{\eta \triangleright s \text{ Stack}}$$

When we call a function, we save the environment to the stack.

$$\overline{(\text{Apply } \langle\langle f.x. e \rangle\rangle \square) \triangleright s \mid \eta \prec v \mapsto_E \eta \triangleright s \mid (x = v, f = \langle\langle f.x. e \rangle\rangle, \eta) \succ e}$$

When the function returns, we restore the old environment, clearing out the new bindings:

$$\overline{\eta \triangleright s \mid \eta' \prec v \mapsto_E s \mid \eta \prec v}$$

This attempt is also broken (we'll see why soon)

Simple Example

○	●	⋈	(Ap (Fun (f.x. (Plus x (N 1)))) (N 3))
(Ap □ (N 3))▷○	●	⋈	(Fun (f.x. (Plus x (N 1))))
(Ap □ (N 3))▷○	●	⋈	⟨⟨f.x. (Plus x (N 1))⟩⟩
(Ap ⟨⟨...⟩⟩ □)▷○	●	⋈	(N 3)
(Ap ⟨⟨...⟩⟩ □)▷○	●	⋈	3
●▷○	x = 3, f = ⟨⟨...⟩⟩, ●	⋈	(Plus x (N 1))
(Plus □ (N 1))▷●▷○	x = 3, f = ⟨⟨...⟩⟩, ●	⋈	x
(Plus □ (N 1))▷●▷○	x = 3, f = ⟨⟨...⟩⟩, ●	⋈	3
(Plus 3 □)▷●▷○	x = 3, f = ⟨⟨...⟩⟩, ●	⋈	(N 1)
(Plus 3 □)▷●▷○	x = 3, f = ⟨⟨...⟩⟩, ●	⋈	1
●▷○	x = 3, f = ⟨⟨...⟩⟩, ●	⋈	4
○	●	⋈	4

Seems

to work for **basic examples**, but is there some way to break it?

Closure Capture

○ | ● \succ (Ap (Ap (Fun ($f.x.$ (Fun ($g.y.$ x)))) (N 3)) (N 4))

\mapsto_E (Ap □ (N 4)) \triangleright ○ | ● \succ (Ap (Fun ($f.x.$ (Fun ($g.y.$ x)))) (N 3))

\mapsto_E (Ap □ (N 3)) \triangleright (Ap □ (N 4)) \triangleright ○ | ● \succ (Fun ($f.x.$ (Fun ($g.y.$ x))))

\mapsto_E (Ap □ (N 3)) \triangleright (Ap □ (N 4)) \triangleright ○ | ● \prec $\langle\langle f.x. (Fun (g.y. x)) \rangle\rangle$

\mapsto_E (Ap $\langle\langle f \dots \rangle\rangle$ □) \triangleright (Ap □ (N 4)) \triangleright ○ | ● \succ (N 3)

\mapsto_E (Ap $\langle\langle f \dots \rangle\rangle$ □) \triangleright (Ap □ (N 4)) \triangleright ○ | ● \prec 3

\mapsto_E ● \triangleright (Ap □ (N 4)) \triangleright ○ | $x = 3, f = \langle\langle f \dots \rangle\rangle, \bullet \succ (Fun (g.y. x))$

\mapsto_E ● \triangleright (Ap □ (N 4)) \triangleright ○ | $x = 3, f = \langle\langle f \dots \rangle\rangle, \bullet \prec \langle\langle g.y. x \rangle\rangle$

\mapsto_E (Ap □ (N 4)) \triangleright ○ | ● $\prec \langle\langle g.y. x \rangle\rangle$

\mapsto_E (Ap $\langle\langle g.y. x \rangle\rangle$ □) \triangleright ○ | ● \succ (N 4)

\mapsto_E (Ap $\langle\langle g.y. x \rangle\rangle$ □) \triangleright ○ | ● \prec 4

\mapsto_E ● \triangleright ○ | $y = 4, g = \langle\langle g.y. x \rangle\rangle, \bullet \succ x$

Oh no! We're stuck!

Something went wrong!

When we return functions, the function's body escapes the scope of bound variables from where it was defined:

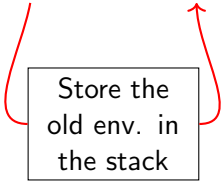
(let $x = 3$ in recfun f $y = x + y$) 5

The function value $\langle\langle f.y. x + y \rangle\rangle$, when it is applied, does not “remember” that $x = 3$.

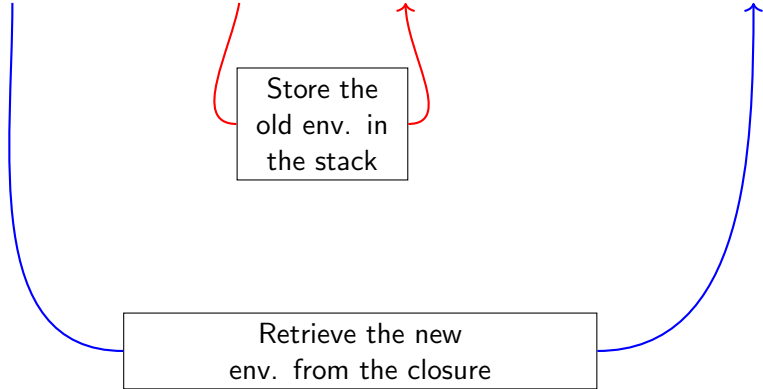
Solution: Store the environment inside the function value!

$$\frac{}{s \mid \eta \succ (\text{Recfun } (f.x. e)) \mapsto_E s \mid \eta \prec \langle\langle \eta, f.x. e \rangle\rangle}$$

This type of function value is called a *closure*.

$$(\text{Apply } \langle\langle \eta', f.x. e \rangle\rangle \square) \triangleright s \mid \eta \prec v \mapsto_E \eta \triangleright s \mid (x = v, f = \langle\langle f.x. e \rangle\rangle, \eta') \succ e$$


Store the
old env. in
the stack



Retrieve the new
env. from the closure

Our Example

$\circ \mid \bullet \succ (\text{Ap } (\text{Ap } (\text{Fun } (f.x. (\text{Fun } (g.y. x)))) (\text{N } 3)) (\text{N } 4))$
 $(\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid \bullet \succ (\text{Ap } (\text{Fun } (f.x. (\text{Fun } (g.y. x)))) (\text{N } 3))$
 $(\text{Ap } \square (\text{N } 3)) \triangleright (\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid \bullet \succ (\text{Fun } (f.x. (\text{Fun } (g.y. x))))$
 $(\text{Ap } \square (\text{N } 3)) \triangleright (\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid \bullet \prec \langle \bullet, f.x. (\text{Fun } (g.y. x)) \rangle$
 $(\text{Ap } \langle \bullet, f \dots \rangle \square) \triangleright (\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid \bullet \succ (\text{N } 3)$
 $(\text{Ap } \langle \bullet, f \dots \rangle \square) \triangleright (\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid \bullet \prec 3$
 $\bullet \triangleright (\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid x = 3, f = \langle f \dots \rangle, \bullet \succ (\text{Fun } (g.y. x))$
 $\bullet \triangleright (\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid x = 3, f = \langle f \dots \rangle, \bullet \prec \langle (x = 3, f = \dots, \bullet), g.y. x \rangle$
 $(\text{Ap } \square (\text{N } 4)) \triangleright \circ \mid \bullet \prec \langle (x = 3, f = \dots, \bullet), g.y. x \rangle$
 $(\text{Ap } \langle (x = 3, f = \dots, \bullet), g.y. x \rangle \square) \triangleright \circ \mid \bullet \succ (\text{N } 4)$
 $(\text{Ap } \langle (x = 3, f = \dots, \bullet), g.y. x \rangle \square) \triangleright \circ \mid \bullet \prec 4$
 $\bullet \triangleright \circ \mid y = 4, g = \langle g.y. x \rangle, x = 3, f = \dots, \bullet \succ x$
 $\bullet \triangleright \circ \mid y = 4, g = \langle g.y. x \rangle, x = 3, f = \dots, \bullet \prec 3$
 $\circ \mid \bullet \prec 3$

Refinement

- We already sketched a proof that each C-machine execution has a corresponding M-machine execution (**refinement**).
- This means any functional correctness (not security or cost) property we prove about all M-machine executions of a program apply just as well to any C-machine executions of the same program.
- Now we want to prove that each E-machine execution has a corresponding C-machine execution (and therefore an M-machine execution).

Ingredients for Refinement

Once again, we want an **abstraction function** \mathcal{A} that converts E-machine states to C-machine states, such that:

- Each initial state in the E-machine is mapped to an initial state in the C-Machine.
- Each final state in the E-machine is mapped to a final state in the C-Machine.
- For each E-machine transition, either there is a corresponding C-Machine transition, or the two E-machine states map to the same C-machine state.

How to define \mathcal{A} ?

- Our abstraction function \mathcal{A} applies the environment η as a **substitution** to the current expression, and to the stack, starting at the left.
- If any environment is encountered in the stack, switch to substituting with that environment instead.
- E-Machine values are converted to C-Machine values merely by applying the environment inside closures as a substitution to the expression inside the closure.

With such a function definition, it is trivial to prove that each E-Machine transition has a corresponding transition in the C-Machine, as it is 1:1.

Except!

There is one rule which is not 1:1. **Which one?**