THE E-MACHINE

A further improvement on the C-machine: function app. no longer executed via Substitutions.

How: add environments $\cong$ a "mapping" variables $\longmapsto$ values.

$\bullet$ — empty

$x=v, \eta$ — nonempty, sets $x=v$ in the environment η

A variable could have many values in a mapping
$\Rightarrow$ we are interested in the latest one, defined to be $\eta(x)$.

C-machine states now include an environment:

$$s \mid \eta > e \quad \text{or} \quad s \mid \eta < e.$$

Also need closures: $\langle\langle \eta, f.x.e \rangle\rangle$

η: env. in which f was def'd

$f.x.e$: function body from a recfun

Why? Avoids function bodies existing in inappropriate scopes (i.e. one in which bound variables are unset / have diff. vals)

Ex. 2: Example where closures are necessary:

$$(\text{let } x=1 \text{ in } (\text{let } x=2 \text{ in } \text{recfun } f\ y = x)))\ 3\ 6\ 1$$

Without closures, get $x=1 \Rightarrow$ result is 1
With closures, we remember the inner $x=2 \Rightarrow$ result is 2

^ which is what we expect to happen!

Ex. 3 (b): Optimise the E-machine for tail recursion with another rule

---

$$(\mathrm{Ap}\ \langle\langle n_f, f.x.e\rangle\rangle\ \square) \triangleright n \triangleright s \mid n' \prec v \longmapsto_E n \triangleright s \mid (x = v, f = \langle\langle f.x.e\rangle\rangle, n_f) \succ e$$

↖ i.e. optimise for env under Ap frame

SAFETY AND LIVENESS PROPERTIES

Trace: sequence of program states

$\sigma_1 \longmapsto \sigma_2 \longmapsto \sigma_3 \longmapsto \ldots \longmapsto \sigma_n$ ↙ final state.

Behaviour: an infinite trace (if finite, just infinitely repeat last state)

Property: A set of behaviours.

While abstract, this def'n has uses in type theory and concurrency

Two fundamental types of property:

Safety	Liveness
Something bad will never happen	Something good will happen
can be violated by a finite prefix of a behaviour	cannot be violated by any finite prefix of any behaviour

Example. Type safety is a safety property.

i.e. any term that has a type does not end up in a stuck state

Termination is liveness.

i.e. The process will halt execution in $< \infty$ steps.

Fact: every property = safety ∩ liveness property. [Alpern, Schneider]

Ex. 4(c): Decompose these properties into safety ∩ liveness:

1. "The program will allocate exactly 100MB of memory"

   S:
   L:

2. "The program will not overflow the stack"

   S:
   L:

3. "The program will return the sum of its inputs"

   S:
   L:

4. "The program will call the function f"

   S:
   L: