Lec 12 Dynamic Semantics

Static semantics : what's a valid programme
Dynamic " : how the programme should be executed
└ English : ambiguous
└ Math : more precise ← WASM defined this way, for example

Types of Dyn. Sem.

- Denotational — abstract, elegant
  programmes ↔ mathematical object
  procedure ↔ function
- Axiomatic — programme logic
  code ↔ proof
- Operational (very common) — how programme is executed
  abstract machines
  - └ Structural operational
  - └ Substructural operational
  - └ Abstract machine
  - ⋮
- ⋮

Continuation

Transition system : step by step eval of expr until reduced to value

$$e \triangleright k$$

(e: expr, k: continuation)

Pure Ops

$$e_1 \oplus e_2 \triangleright K \rightarrow e_1 \triangleright (\_ \oplus e_2, k)$$
$$c_1 \oplus e_2 \triangleright K \rightarrow e_2 \triangleright (c_1 \oplus \_, k) \quad c_1 \text{ is const}$$
$$c_2 \triangleright (c_1 \oplus \_, k) \rightarrow c \triangleright K \quad c = c_1 + c_2 \pmod{2^{32}}$$

Fx Ops

$$e_1 \oslash e_2 \triangleright K \rightarrow e_1 \triangleright (\_ \oslash e_2, k)$$
$$c_1 \oslash e_2 \triangleright K \rightarrow e_2 \triangleright (c_1 \oslash \_, k)$$
$$c_2 \triangleright (c_1 \oslash \_, k) \rightarrow c \triangleright K \quad c = c_1 \oslash c_2$$
$$c_2 \triangleright (c_1 \oslash \_, k) \rightarrow \text{exception(arith)} \quad c_1 \oslash c_2 \text{ undefined}$$

Eval ex.

$$((4+5)*10)+2 \triangleright \cdot$$
$$\rightarrow (4+5)*10 \triangleright \_ + 2$$
$$\rightarrow 4+5 \triangleright \_ * 10, \_ + 2$$
$$\rightarrow 4 \triangleright \_ + 5, \_ * 10, \_ + 2$$
$$\rightarrow 5 \triangleright 4 + \_, \_ * 10, \_ + 2$$
$$\rightarrow \ldots$$

Bool Ops

$$e_1 \,\&\&\, e_2 \triangleright K \rightarrow e_1 \triangleright (\_ \,\&\&\, e_2, k)$$
$$\text{false} \triangleright (\_ \,\&\&\, e_2, k) \rightarrow \text{false} \triangleright k \quad \leftarrow \text{shortcutting}$$
$$\text{true} \triangleright (\_ \,\&\&\, e_2, k) \rightarrow e_2 \triangleright k$$

Variables

define η as environment $\quad \eta ::= \cdot \mid \eta, x \to v$

$$\eta \vdash x \triangleright K \rightarrow \eta \vdash \eta(x) \triangleright K \quad x \text{ required to be in } \eta$$

Statement Conts

statement doesn't pass value to k but may modify η

$$\eta \vdash s \triangleright K$$
$$\eta \vdash \text{seq}(s_1, s_2) \triangleright K \rightarrow \eta \vdash s_1 \triangleright (s_2, k)$$
$$\eta \vdash \text{nop} \triangleright (s, k) \rightarrow \eta \vdash s \triangleright k$$

$$\eta \vdash \text{assign}(x, e) \triangleright K \rightarrow \eta \vdash e \triangleright (\text{assign}(x, \_), k)$$
$$\eta \vdash v \triangleright (\text{assign}(x, \_), k) \rightarrow \eta[x \mapsto v] \vdash \text{nop} \triangleright k$$

$$\eta \vdash \text{if}(e, s_1, s_2) \triangleright K \rightarrow \eta \vdash e \triangleright (\text{if}(\_, s_1, s_2), k)$$
$$\eta \vdash \text{true} \triangleright (\text{if}(\_, s_1, s_2), k) \rightarrow \eta \vdash s_1 \triangleright K$$
$$\eta \vdash \text{false} \triangleright (\text{if}(\_, s_1, s_2), k) \rightarrow \eta \vdash s_2 \triangleright K$$

Observe $\text{while}(e, s) \equiv \text{if}(e, \text{seq}(s, \text{while}(e, s)), \text{nop})$

$$\eta \vdash \text{while}(e, s) \triangleright K \rightarrow \eta \vdash \text{if}(e, \text{seq}(s, \text{while}(e, s)), \text{nop}) \triangleright K$$

Declaration

$$\eta \vdash \text{decl}(x, \tau, s) \triangleright K \rightarrow \eta[x \mapsto \text{nothing}] \vdash s \triangleright K$$

(No shadowing in CO so this is fine)

Assertions

$$\eta \vdash \text{assert}(e) \triangleright K \rightarrow \eta \vdash e \triangleright (\text{assert}(\_), K)$$
$$\eta \vdash \text{true} \triangleright (\text{assert}(\_), k) \rightarrow \eta \vdash \text{nop} \triangleright k$$
$$\eta \vdash \text{true} \triangleright (\text{assert}(\_), k) \rightarrow \text{exception(abort)}$$

Final state

$$\text{exception}(E)$$
$$\text{nop} \triangleright \cdot$$

Functions Cont.

eval the arguments, save caller env, save caller cont., run callee, restore stuff

Call stack $\quad S ::= \cdot \mid S, \langle \eta, K \rangle$

$$S; \eta \vdash f() \triangleright K \rightarrow (S, \langle \eta, k \rangle); \cdot \vdash s \triangleright \cdot$$
$$S; \eta \vdash f(e_1, e_2) \triangleright k \rightarrow S; \eta \vdash e_1 \triangleright (f(\_, e_2), k)$$
$$S; \eta \vdash c_1 \triangleright (f(\_, e_2), k) \rightarrow S; \eta \vdash e_2 \triangleright (f(c_1, \_), k)$$
$$\vdots$$

Return

$$S; \eta \vdash \text{return}(e) \triangleright K \rightarrow S; \eta \vdash e_1 \triangleright (\text{return}(\_), k)$$
$$S, \langle \eta', k' \rangle; \eta \vdash v \triangleright (\text{return}(\_), k) \rightarrow S; \eta' \vdash v \triangleright K'$$

Special case : return void

$$S, \langle \eta', k' \rangle; \eta \vdash \text{nop} \triangleright \cdot \rightarrow S; \eta' \vdash \text{nothing} \triangleright k'$$

(nothing: dummy)

Main function (initial state)

$$\cdot; \cdot \vdash \text{main}() \triangleright \cdot$$

Expression statement

$$\eta \vdash e \triangleright k \rightarrow \eta \vdash e \triangleright (\text{discard}, k)$$
$$\eta \vdash v \triangleright (\text{discard}, k) \rightarrow \eta \vdash \text{nop} \triangleright K$$

(v would be nothing if passing void function)

Formal stuff

Thm if programme passes static analysis, we shouldn't get stuck executing by those rules