Meeting 26: Procedural Abstraction

Today
Parameter Passing

Commentary
- Things are "clicky"
- We are writing programs that take programs as input
  - very "higher-order"

Type Checking
Step-Step Deref

1. Mutable Variables
2. Dynamic Memory Allocation
3. Parameter Passing Modes
def plus(x: Int, y: Int): Int =

fun body \( x + y \) (formal parameter)

plus(3, 4)
plus(15, 6)

On a fun call, we "replace" the actual argument for the formal parameter in the fun body

Parameter passing - interaction between the actuals & formals
Usual: pass the value from caller to callee

"Exotic" output / input

out param - pass into from callee to caller

Don't really need out params because you have return values
int f() { return 42; }  // call
x = f();

--- vs. ---

void f(out int r) { r = 42; }  // call
f(x)

void f(out int r, out int s) { r = 1; s = 2; }  // call
f(x, x)

"When is the address determined for copying back?"

callee has local copy + results are "copied back" on return

caller / callee share a cell
\[ a_2 \in \text{dom } (M) \]

\[ \langle M, (\text{ref } x : z \Rightarrow e_1)(*a_2) \rangle \rightarrow \langle M, [\ast a_2/x]e_1 \rangle \]

This is the "Do" rule for call-by-referring.

The cell being shared is the cell with address \(a_2\).

\[ (\text{ref } x : z \Rightarrow x = 1) \ (3) \]

Type error

Need cell here
Eager Evaluation = evaluate actual argument to a value before making the function call

Mutation

- call-by-value
- call-by-ref
- call-by-value- "result"
- copy

Lazy Evaluation

- call-by-name

"being lazy"

- don't evaluate argument because the function may not use the argument
\( (\text{name } x : x) \Rightarrow e_1)(e_2) \rightarrow [e_2/x]e_1 \)

call-by-name  lazy evaluation