CS 330 Lecture ?

- Operational Semantics for uscheme

Variables & functions

Variables refer to locations – to examine or change them
1) lookup their names in p (the environment) to find their locations
2) look or change the contents of the lcoations

Let-bindings introduce fresh locations, bind them to variables and initialize them

Function application also introduces fresh locations, which hold actual parameters. These lcoations are then bound to the names of the formal parameters of the function being applied.

Variables

$$\frac{x \in \mathrm{dom}\ p \qquad p(x) \in \mathrm{dom}\ s}{\langle \mathrm{VAR}(x), p, s \rangle \Rightarrow \langle s(p(x)), s \rangle}$$

$$\frac{x \in \mathrm{dom}\ p \qquad p(x) = l \qquad \langle e, p, s \rangle \Rightarrow \langle v, s' \rangle}{\langle \mathrm{SET}(x,e), p, s \rangle \Rightarrow \langle v, s'\{l \to v\} \rangle}$$

lookup of variable
first find location
to which x corresponds
then find value in that
location
(compiled time
runtime)

Function abstraction & application

Functional abstraction wraps the current environment, along with a lambda expression in a closure. LAMBDA makes a copy of the current environment. Because environments can be copied, they have to map names to locations, not values in order to supprort shared mutable state.

Function application uses the environment in the closure, extended by binding the fomral parameters to fresh locations. These locations are initialized by the values of the actual parameters, but the body e might change the contents of these locations during it's evaluation.

Function Application II

Environment in a closure, binds the free variables of the closure (no such environment is needed in Impcore, because all functions are defined at top level, so any free variables can necessarily be found in the global environment g)

Parameters don't extend the empty environment as in ImpCore but the environment Pc stored in the closures.

IMPORTANT: the evaluation is completely independent of the environment p of the calling function, a Scheme (or ML) function behaves the same way no matter where it is called from

Understanding static vs dynamic scoping

In Scheme each function is <lambda expression, environment>
In Lisp each function is <lambda expression>

How do we get the free variables ?
From the environment of the caller (dynamic scoping)