Lecture #19: Operational Semantics, Continued

Operations on Lists: Concatenation

- List concatenation again requires allocation:

$$G, E, S_0 \vdash e_1 : v_1, S_1, \_$$
$$G, E, S_1 \vdash e_2 : v_2, S_2, \_$$
$$v_1 = [l_1, l_2, \ldots, l_n]$$
$$v_2 = [l'_1, l'_2, \ldots, l'_m]$$
$$n, m \geq 0$$
$$l''_1, \ldots, l''_{m+n} = newloc(S_2, m+n)$$
$$v_3 = [l''_1, l''_2, \ldots, l''_{n+m}]$$
$$\frac{S_3 = S_2[S_2(l_1)/l''_1] \ldots [S_2(l_n)/l''_n][S_2(l'_1)/l''_{n+1}] \ldots [S_2(l'_m)/l''_{n+m}]}{G, E, S_0 \vdash e_1 + e_2 : v_3, S_3, \_} \quad [\text{LIST-CONCAT}]$$

- Subtlety here: in $e_1 + e_2$, suppose evaluating e_2 has a side-effect on e_1. What value goes into the resulting list?

Classes and Functions as Values

- Although the language does not allow programmers to treat classes as first-class values, the semantics is free to do so.

- Thus, we represent a class T with attributes $a_1, \ldots$ a

$$class(T) = (a_1 = e_1, \ldots, a_m = e_m)$$

- Here, the e_i are either literal expressions or function definitions.

- Similarly, the semantics associates a function value with each function, even though programmers cannot manipulate functions as values:

$$v = (x_1, \ldots, x_n, y_1 = e_1, \ldots, y_k = e_k, b_{body}, E_f)$$

- Here, the x_i are parameter names and the y_i are the local variables (with initializers e_i.)

Function Invocation

Now things get complicated:

$$\frac{S_0(E(f)) = (x_1, \ldots, x_n, y_1 = e'_1, \ldots, y_k = e'_k, b_{body}, E_f)}{n, k \geq 0}$$ Evaluate Function

$$G, E, S_0 \vdash e_1 : v_1, S_1, \_$$ Evaluate Actuals

$$\vdots$$

$$\frac{G, E, S_{n-1} \vdash e_n : v_n, S_n, \_}{\begin{array}{c} l_{x1}, \ldots, l_{xn}, l_{y1}, \ldots, l_{yk} = newloc(S_n, n+k) \\ E' = E_f[l_{x1}/x_1] \ldots [l_{xn}/x_n][l_{y1}/y_1] \ldots [l_{yk}/y_k] \end{array}}$$ Allocate New Locations for Parameters and Locals

$$G, E', S_n \vdash e'_1 : v'_1, S_n, \_$$

$$\vdots$$ Evaluate Initializers

$$\frac{G, E', S_n \vdash e'_k : v'_k, S_n, \_}{S_{n+1} = S_n[v_1/l_{x1}] \ldots [v_n/l_{xn}][v'_1/l_{y1}] \ldots [v'_k/l_{yk}]}$$ Assign Params. and Locals

$$G, E', S_{n+1} \vdash b_{body} : \_, S_{n+2}, R$$ Evaluate Body

$$R' = \begin{cases} None, & \text{if } R \text{ is } \_ \\ R, & \text{otherwise} \end{cases}$$ And Capture Return Value

$$G, E, S_0 \vdash f(e_1, \ldots, e_n) : R', S_{n+2}, \_$$ [INVOKE]

Function Invocation: Discussion

- In the rules for evaluating local definitions:

$$G, E', S_n \vdash e'_1 : v'_1, S_{n,} \_$$
$$\vdots$$
$$G, E', S_n \vdash e'_k : v'_k, S_{n,} \_$$

Why does the state remain S_n?

- What would happen if we changed the steps for allocating new variables to

$$\frac{\begin{array}{c} G, E, S_n \vdash e'_1 : v'_1, S_{n,} \_ \\ \vdots \\ G, E, S_n \vdash e'_k : v'_k, S_{n,} \_ \end{array}}{\begin{array}{l} l_{x1}, \ldots, l_{xn}, l_{y1}, \ldots, l_{yk} = newloc(S_n, n+k) \\ E' = E_f[l_{x1}/x_1] \ldots [l_{xn}/x_n][l_{y1}/y_1] \ldots [l_{yk}/y_k] \end{array}}$$

  Evaluate Initializers

  Allocate New Locations for Parameters and Locals

?

Function Invocation: Discussion

- In the rules for evaluating local definitions:

$$G, E', S_n \vdash e'_1 : v'_1, S_n, \rule{1em}{0.08em}$$
$$\vdots$$
$$G, E', S_n \vdash e'_k : v'_k, S_n, \rule{1em}{0.08em}$$

Why does the state remain S_n? The e'_i must all be literals or function definitions, whose evaluation does not change the state.

- What would happen if we changed the steps for allocating new variables to

$$G, E, S_n \vdash e'_1 : v'_1, S_n, \rule{1em}{0.08em}$$
$$\vdots \qquad \qquad \text{Evaluate Initializers}$$
$$G, E, S_n \vdash e'_k : v'_k, S_n, \rule{1em}{0.08em}$$

$$l_{x1}, \ldots, l_{xn}, l_{y1}, \ldots, l_{yk} = newloc(S_n, n + k) \quad \text{Allocate New Locations for}$$
$$E' = E_f[l_{x1}/x_1] \ldots [l_{xn}/x_n][l_{y1}/y_1] \ldots [l_{yk}/y_k] \quad \text{Parameters and Locals}$$

?

Function Invocation: Discussion

- In the rules for evaluating local definitions:

$$G, E', S_n \vdash e'_1 : v'_1, S_n, \rule{0.6cm}{0.4pt}$$
$$\vdots$$
$$G, E', S_n \vdash e'_k : v'_k, S_n, \rule{0.6cm}{0.4pt}$$

Why does the state remain S_n? The e'_i must all be literals or function definitions, whose evaluation does not change the state.

- What would happen if we changed the steps for allocating new variables to

$$\frac{\begin{array}{l} G, E, S_n \vdash e'_1 : v'_1, S_n, \rule{0.6cm}{0.4pt} \\ \qquad\qquad \vdots \qquad\qquad\qquad\qquad \text{Evaluate Initializers} \\ G, E, S_n \vdash e'_k : v'_k, S_n, \rule{0.6cm}{0.4pt} \end{array}}{\begin{array}{l} l_{x1}, \ldots, l_{xn}, l_{y1}, \ldots, l_{yk} = newloc(S_n, n+k) \quad \text{Allocate New Locations for} \\ E' = E_f[l_{x1}/x_1] \ldots [l_{xn}/x_n][l_{y1}/y_1] \ldots [l_{yk}/y_k] \quad \text{Parameters and Locals} \end{array}}$$

? Nothing. The effect is the same.

Function Invocation: Discussion (II)

- Consider the lines:

$$S_0(E(f)) = (x_1, \ldots, x_n, y_1 = e'_1, \ldots, y_k = e'_k, b_{body}, E_f) \quad \text{Evaluate Function}$$

$$\vdots$$

$$E' = E_f[l_{x1}/x_1] \ldots [l_{xn}/x_n][l_{y1}/y_1] \ldots [l_{yk}/y_k] \quad \text{Parameters and Locals}$$

$$\vdots$$

$$G, E', S_{n+1} \vdash b_{body} : \_\_, S_{n+2}, R \quad \text{Evaluate Body} \enspace \vdots$$

$$\overline{G, E, S_0 \vdash f(e_1, \ldots, e_n) : R', S_{n+2}, \_\_ \qquad [\text{INVOKE}]}$$

- The environment for evaluating the body, E', is *not* an extension of E, but rather of E_f, the environment that is part of the function's value.

- This is in keeping with the rule you first saw in CS61A: a function value's parent frame is the one in which the function definition is evalated, not the one in which the call is evaluated.

Method Dispatching

Method dispatching, as in `x.f(3)`, is unsurprisingly close to function invocation.

$$\frac{
\begin{array}{l}
G, E, S \vdash e_0 : v_0, S_0, \_ \quad \text{(Evaluate Object)} \\
v_0 = X(a_1 = l_1, \ldots, f = l_f, \ldots, a_m = l_m) \quad \text{(Find f in Class Value)} \\
S_0(l_f) = (x_0, x_1, \ldots, x_n, y_1 = e'_1, \ldots, y_k = e'_k, b_{body}, E_f) \\
\hline
\ldots \textit{Evaluate Parameters as for Function Calls} \ldots \\
\hline
l_{x0}, l_{x1}, \ldots, l_{xn}, l_{y1}, \ldots, l_{yk} = newloc(S_n, n+k+1) \\
E' = E_f[l_{x0}/x_0] \ldots [l_{xn}/x_n][l_{y1}/y_1] \ldots [l_{yk}/y_k] \\
\hline
\ldots \textit{Evaluate Initializers for Locals} \\
\textit{as for Function Calls} \ldots \\
\hline
S_{n+1} = S_n[v_0/l_{x0}] \ldots [v_n/l_{xn}][v'_1/l_{y1}] \ldots [v'_k/l_{yk}] \quad \text{(Assign Params. and Locals)} \\
\hline
G, E', S_{n+1} \vdash b_{body} : \_, S_{n+2}, R \quad \text{(Evaluate Body)} \\
R' = \begin{cases} None, & \text{if } R \text{ is } \_ \\ R, & \text{otherwise} \end{cases} \quad \text{(And Capture Return Value)}
\end{array}
}{
G, E, S \vdash e_0.f(e_1, \ldots, e_n) : R', S_{n+2}, \_ \qquad [\text{DISPATCH}]
}$$

Evaluate Object
Find f in Class Value
... Evaluate Parameters as for Function Calls ...
... Evaluate Initializers for Locals as for Function Calls ...
Assign Params. and Locals
Evaluate Body
And Capture Return Value
[DISPATCH]

Function Definitions

- Function definitions provide values for local definitions (nested functions) and global functions.

- In the $[\textsc{invoke}]$ and $[\textsc{dispatch}]$ rules, these values then get assigned to the local names (denoted y_i in those rules).

$$\frac{\begin{array}{c} g_1, \ldots, g_L : \text{ variables declared with '}\mathbf{global}\text{' in } f \\ y_1 = e_1, \ldots, y_k = e_k : \text{ local variables and functions in } f \\ E_f = E[G(g_1)/g_1] \ldots [G(g_L)/g_L] \\ v = (x_1, \ldots, x_n, y_1 = e_1, \ldots, y_k = e_k, b_{body}, E_f) \end{array}}{G, E, S \vdash \texttt{def } f(x_1\!:\!T_1, \ldots, x_n\!:\!T_n) \ [\![\texttt{-> } T_0]\!]^? : b : v, S, \_} \ [\textsc{func-method-def}]$$

- This is where we capture the parent local environment, E, in which f is defined.

Native Functions

- Certain functions are predefined (`print`, `len`, `input`), and do not have normal bodies.

- For these, we denote the function bodies as, e.g., `native len` and define special rules for these particular bodies.

- Assume that the native bodies expect a parameter named `val` (if they have one).

- Then we can define, e.g.,

$$\frac{S(E(\texttt{val})) = v \qquad v = int(i) \textbf{ or } v = bool(b) \textbf{ or } v = str(n,s)}{G, E, S \vdash \texttt{native print} : \_, S, None} \; \big[\text{PRINT}\big]$$

$$\frac{S(E(\texttt{val})) = v \qquad v = [l_1, l_2, \ldots, l_n] \qquad n \geq 0}{G, E, S \vdash \texttt{native len} : \_, S, int(n)} \; \big[\text{LEN-LIST}\big]$$

and others.

Accessing Attributes of Classes

The notation from the first slide provides us with a description of a value and its attributes:

$$\frac{\begin{array}{c} G, E, S_0 \vdash e : v_1, S_1, \text{---} \\ v_1 = X(a_1 = l_1, \ldots, id = l_{id}, \ldots, a_m = l_m) \\ v_2 = S_1(l_{id}) \end{array}}{G, E, S_0 \vdash e.id : v_2, S_1, \text{---}} \quad [\text{ATTR-READ}]$$

$$\frac{\begin{array}{c} G, E, S_0 \vdash e_2 : v_r, S_1, \text{---} \\ G, E, S_1 \vdash e_1 : v_l, S_2, \text{---} \\ v_l = X(a_1 = l_1, \ldots, id = l_{id}, \ldots, a_m = l_m) \\ S_3 = S_2[v_r/l_{id}] \end{array}}{G, E, S_0 \vdash e_1.id = e_2 : \text{---}, S_3, \text{---}} \quad [\text{ATTR-ASSIGN-STMT}]$$

Q: In [ATTR-ASSIGN-STMT], what exactly happens when e_1 or e_2 have side effects?

Accessing Attributes of Classes

The notation from the first slide provides us with a description of a value and its attributes:

$$\frac{\begin{array}{c} G, E, S_0 \vdash e : v_1, S_1, \text{\textbf{---}} \\ v_1 = X(a_1 = l_1, \ldots, id = l_{id}, \ldots, a_m = l_m) \\ v_2 = S_1(l_{id}) \end{array}}{G, E, S_0 \vdash e.id : v_2, S_1, \text{\textbf{---}}} \quad [\text{ATTR-READ}]$$

$$\frac{\begin{array}{c} G, E, S_0 \vdash e_2 : v_r, S_1, \text{\textbf{---}} \\ G, E, S_1 \vdash e_1 : v_l, S_2, \text{\textbf{---}} \\ v_l = X(a_1 = l_1, \ldots, id = l_{id}, \ldots, a_m = l_m) \\ S_3 = S_2[v_r/l_{id}] \end{array}}{G, E, S_0 \vdash e_1.id = e_2 : \text{\textbf{---}}, S_3, \text{\textbf{---}}} \quad [\text{ATTR-ASSIGN-STMT}]$$

Q: In $[\text{ATTR-ASSIGN-STMT}]$, what exactly happens when e_1 or e_2 have side effects? **A:** e_2 is evaluated first, and therefore can affect the evaluation of e_1, but not vice-versa.

Creating Objects

$$class(T) = (a_1 = e_1, \ldots, a_m = e_m) \qquad \text{\textit{T} Must Be A Class}$$
$$m \geq 1$$
$$l_{a1}, \ldots, l_{am} = newloc(S, m) \qquad \text{Allocate Attributes}$$
$$v_0 = T(a_1 = l_{ai}, \ldots, a_m = l_{am}) \qquad \text{New Object Value}$$

$$\overline{G, G, S \vdash e_1 : v_1, S, \_}$$
$$\vdots \qquad \text{Evaluate Initializers in } G$$
$$G, G, S \vdash e_m : v_m, S, \_$$
$$S_1 = S[v_1/l_{a1}] \ldots [v_m/l_{am}] \qquad \text{Initialize Attributes}$$

$$\overline{l_{init} = l_{ai} \text{ such that } a_i = \texttt{\_\_init\_\_}} \qquad \text{Get } \texttt{\_\_init\_\_} \text{ method}$$
$$S_1(l_{init}) = (x_0, y_1 = e_1', \ldots, y_k = e_k', b_{body}, E_f)$$
$$k \geq 0$$

$$l_{x0}, l_{y1}, \ldots, l_{yk} = newloc(S_1, k+1)$$
$$E' = E_f[l_{x0}/x_0][l_{y1}/y_1] \ldots [l_{yk}/y_k]$$
$$G, E, S_1 \vdash e_1' : v_1', S_1, \_$$
$$\vdots \qquad \text{Call It On } v_0$$
$$G, E, S_1 \vdash e_k' : v_k', S_1, \_$$
$$S_2 = S_1[v_0/l_{x0}][v_1'/l_{y1}] \ldots [v_k'/l_{yk}]$$
$$G, E', S_2 \vdash b_{body} : \_, S_3, \_$$
$$\overline{G, E, S \vdash T() : v_0, S_3, \_}$$

Starting Things Off: Programs

We start with an initial store and environment, containing just the pre-defined function.

$$g_1 = e_1, \ldots, g_k = e_k \text{ are the global variable and}$$
$$\text{function definitions in the program}$$
$$P \text{ is the sequence of statements in the program}$$

$$\frac{\begin{array}{c} l_{g1}, \ldots, l_{gk} = newloc(S_{init}, k) \\ G = G_{init}[l_{g1}/g_1] \ldots [l_{gk}/g_k] \\ G, G, S_{init} \vdash e_1 : v_1, S_{init}, \_ \\ \vdots \\ G, G, \emptyset \vdash e_k : v_k, S_{init}, \_ \\ S = S_{init}[v_k/l_{g1}] \ldots [v_k/l_{gk}] \\ G, G, S \vdash P : \_, S', \_ \end{array}}{\emptyset, \emptyset, \emptyset \vdash P : \_, S', \_} \quad [\text{PROGRAM}]$$

Initial Store and Environment

Here, we use the native body notation from previously.

$$G_{init} = \emptyset[l_{len}/len][l_{print}/print][l_{input}/input]$$

$$S_{init}(l_{print}) = (\texttt{val}, \texttt{native print}, \emptyset)$$

$$S_{init}(l_{len}) = (\texttt{val}, \texttt{native len}, \emptyset)$$

$$S_{init}(l_{input}) = (\texttt{native print}, \emptyset)$$