CMSC 330: Organization of Programming Languages

Operational Semantics Examples

Practice Examples

- Give a derivation that proves the following (where • is the empty environment)
  - •; let x = 5 in let y = 7 in x+y ⇒ 12
  - •; let x = let x = 5 in x+2 in x+2 ⇒ 9
  - •; let f = fun x → x+5 in f 7 ⇒ 12
  - •; let y = 5 in let f = fun x → x+y in let y = 6 in f 7 ⇒ 12
- Using the dynamic scoping version of the function application rule, prove
  - •; let y = 5 in let f = fun x → x+y in let y = 6 in f 7 ⇒ 13

Notation, Operations on Environments

- • is the empty environment (undefined for all ids)
- x:v is the environment that maps x to v and is undefined for all other ids
- If A and A' are environments then A, A' is the environment defined as follows

$$(A, A')(id) = \begin{cases} A'(id) & \text{if } A'(id) \text{ defined} \\ A(id) & \text{if } A'(id) \text{ undefined but } A(id) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

- Idea: A' "overwrites" definitions in A
- For brevity, can write •, A as just A

Rule for Identifiers

A(x) = v
A; x ⇒ v

- x is an identifier
- To determine its value v "look it up" in A!

SOS Rules: Built-in Functions

- For arithmetic, comparison operations, etc.

$$\frac{A; E_1 \Rightarrow v_1 \qquad A; E_2 \Rightarrow v_2}{A; E_1 \text{ op } E_2 \Rightarrow v_1 \text{ op } v_2}$$

- For : :

$$\frac{A; E_1 \Rightarrow v_1 \qquad A; E_2 \Rightarrow v_2}{A; E_1 \text{ : : } E_2 \Rightarrow \langle v_1, v_2 \rangle}$$

- Rules are recursive
- :: is implemented using pairing
  - Type system guarantees result is list

Trees of Semantic Rules

- When we apply rules to an expression, we actually get a tree
  - Corresponds to the recursive evaluation procedure
    - For example: (3 + 4) + 5

$$\dfrac{\dfrac{\bullet; 3 \Rightarrow 3 \qquad \bullet; 4 \Rightarrow 4}{\bullet; (3 + 4) \Rightarrow 7} \qquad \bullet; 5 \Rightarrow 5}{\bullet; (3 + 4) + 5 \Rightarrow 12}$$

Rule for Let Binding

- We evaluate the first expression, and then evaluate the second expression in an environment extended to include a binding for x

$$\frac{A;\ E_1 \Rightarrow v_1 \qquad A,\ x{:}v_1;\ E_2 \Rightarrow v_2}{A;\ \texttt{let}\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v_2}$$

Example: let x = 1 in x+2

$$\frac{A(x) = v}{A;\ x \Rightarrow v}$$

$$\frac{A;\ E_1 \Rightarrow v_1 \qquad A;\ E_2 \Rightarrow v_2}{A;\ E_1\ \text{op}\ E_2 \Rightarrow v_1\ \text{op}\ v_2}$$

$$\frac{A;\ E_1 \Rightarrow v_1 \qquad A,\ x{:}v_1;\ E_2 \Rightarrow v_2}{A;\ \texttt{let}\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v_2}$$

$$\frac{\dfrac{\bullet,x{:}1;\ x \Rightarrow 1 \qquad \bullet,x{:}1;\ 2 \Rightarrow 2}{\bullet;\ 1 \Rightarrow 1} \qquad \bullet,x{:}1;\ x+2 \Rightarrow 3}{\bullet;\ \text{let } x = 1 \text{ in } x+2 \Rightarrow 3}$$

Example: let x = 5 in let y = 7 in x+y

$$\dfrac{\dfrac{\dfrac{}{\bullet; 5 \Rightarrow 5} \quad \dfrac{\dfrac{}{\bullet,x{:}5; 7 \Rightarrow 7} \quad \dfrac{}{\bullet,x{:}5,\ y{:}7; x+y \Rightarrow 12}}{\bullet,x{:}5; \text{let } y = 7 \text{ in } x+y \Rightarrow 12}}{\bullet; \text{let } x = 5 \text{ in let } y = 7 \text{ in } x+y \Rightarrow 12}$$

Example: let x = let x = 5 in x+2 in x+2

$$\dfrac{\dfrac{\dfrac{}{\bullet,x{:}5; x \Rightarrow 5} \quad \dfrac{}{\bullet,x{:}5; 2 \Rightarrow 2}}{\bullet; 5 \Rightarrow 5} \quad \dfrac{\dfrac{}{\bullet,x{:}5; x+2 \Rightarrow 7} \quad \dfrac{\dfrac{}{\bullet,x{:}7; x \Rightarrow 7} \quad \dfrac{}{\bullet,x{:}7; 2 \Rightarrow 2}}{\bullet,x{:}7; x+2 \Rightarrow 9}}{\bullet; \text{let } x = 5 \text{ in } x+2 \Rightarrow 7}}{\bullet; \text{let } x = \text{let } x = 5 \text{ in } x+2 \text{ in } x+2 \Rightarrow 9}$$

Rule for Function Definitions

$$\frac{}{A;\ \texttt{fun}\ x \rightarrow E\ \Rightarrow (A, \lambda x.E)}$$

- The expression evaluates to a closure
  - The id and body in the closure come from the expression
  - The environment is the one in effect when the expression is evaluated
- This will be used to implement static scope

Rule for Function Application

$$\frac{A;\ E_1 \Rightarrow (A', \lambda x.E) \quad A;\ E_2 \Rightarrow v_2 \quad A', x{:}v_2;\ E \Rightarrow v}{A;\ E_1\ E_2 \Rightarrow v}$$

- 1st hypothesis: E_1 evaluates to a closure
- 2nd hypothesis: E_2 produces a value (call by value!)
- 3rd hypothesis: Body E in modified closure environment produces a value
- This last value is the result of the application

Example: (fun x → x + 3) 4

$$\frac{\bullet, x{:}4;\ x \Rightarrow 4 \qquad \bullet, x{:}4;\ 3 \Rightarrow 3}{\begin{array}{c}\bullet;\ \text{fun } x \to x + 3 \Rightarrow (\bullet, \lambda x.x + 3) \\ \bullet;\ 4 \Rightarrow 4 \\ \bullet, x{:}4;\ x + 3 \Rightarrow 7\end{array}}{\bullet;\ (\text{fun } x \to x + 3)\ \ 4 \Rightarrow 7}$$

Example: (fun x → (fun y → x + y)) 3 4

$$\frac{\bullet;\ (\text{fun } x \to (\text{fun } y \to x + y)) \Rightarrow (\bullet, \lambda x.(\text{fun } y \to x + y)) \qquad \bullet;\ 3 \Rightarrow 3 \qquad x{:}3;\ (\text{fun } y \to x + y) \Rightarrow (x{:}3, \lambda y.(x + y))}{\bullet;\ (\text{fun } x \to (\text{fun } y \to x + y))\ \ 3 \Rightarrow (x{:}3, \lambda y.(x + y))}$$

Let <previous> = (fun x → (fun y → x + y)) 3

Example (cont.)

$$\frac{\dfrac{\bullet, x:3, y:4;\ x \Rightarrow 3 \qquad \bullet, x:3, y:4;\ y \Rightarrow 4}{\bullet;\ \langle previous \rangle \Rightarrow (x:3, \lambda y.(x + y)) \qquad \bullet;\ 4 \Rightarrow 4 \qquad x:3, y:4;\ (x + y) \Rightarrow 7}}{\bullet;\ (\ \langle previous \rangle\ 4\ ) \Rightarrow 7}$$

Example: let f = fun x → x+5 in f 7

$$\frac{\dfrac{\dfrac{\bullet, x:7;\ x \Rightarrow 7 \qquad \bullet, x:7;\ 5 \Rightarrow 5}{\bullet, f:(\bullet, \lambda x.x+5);\ f \Rightarrow (\bullet, \lambda x.x+5) \qquad \bullet, f:(\bullet, \lambda x.x+5);\ 7 \Rightarrow 7 \qquad \bullet, x:7;\ x+5 \Rightarrow 12} \qquad \bullet;\ \text{fun } x \rightarrow x+5 \Rightarrow (\bullet, \lambda x.x+5) \qquad \bullet, f:(\bullet, \lambda x.x+5);\ f\ 7 \Rightarrow 12}{\bullet;\ \text{let f = fun x} \rightarrow \text{x+5 in f 7} \Rightarrow 12}$$

Example: let y = 5 in let f = fun x → x+y in let y = 6 in f 7

Static scoping uses environment from closure

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\bullet;\ 5 \Rightarrow 5}\quad \dfrac{}{y{:}5,\ x{:}7;\ x \Rightarrow 7} \quad \dfrac{}{y{:}5,\ x{:}7;\ y \Rightarrow 5}}{y{:}5,\ f{:}(y{:}5,\ \lambda x.x+y),\ y{:}6;\ f \Rightarrow (y{:}5,\ \lambda x.x+y)} \quad \dfrac{}{y{:}5,\ f{:}(y{:}5,\ \lambda x.x+y),\ y{:}6;\ 7 \Rightarrow 7} \quad \dfrac{}{y{:}5,\ x{:}7;\ x+y \Rightarrow 12}}{y{:}5,\ f{:}(y{:}5,\ \lambda x.x+y),\ y{:}6;\ f\ 7 \Rightarrow 12}}{y{:}5,\ f{:}(y{:}5,\ \lambda x.x+y);\ 6 \Rightarrow 6} \quad \dfrac{}{y{:}5,\ f{:}(y{:}5,\ \lambda x.x+y);\ \text{let } y = 6 \text{ in } f\ 7 \Rightarrow 12}}{y{:}5;\ \text{fun } x \rightarrow x+y \Rightarrow (y{:}5,\ \lambda x.x+y)} \quad \dfrac{}{y{:}5;\ \text{let } f = \text{fun } x \rightarrow x+y \text{ in } \ldots \text{ in } f\ 7 \Rightarrow 12}}{\bullet;\ \text{let } y = 5 \text{ in let } f = \text{fun } x \rightarrow x+y \text{ in let } y = 6 \text{ in } f\ 7 \Rightarrow 12}$$

Dynamic Scoping

- The previous rule for functions implements static scoping, since it implements closures
- We could easily implement dynamic scoping

$$\dfrac{A;\ E_1 \Rightarrow (A',\ \lambda x.E) \quad A;\ E_2 \Rightarrow v_2 \quad \mathbf{A},\ x{:}v_2;\ E \Rightarrow v}{A;\ E_1\ E_2 \Rightarrow v}$$

- The only difference is to use the current environment A, not the environment A'
  - Easy to see the origins of the dynamic scoping bug!

Example: let y = 5 in let f = fun x → x+y in let y = 6 in f 7

Dynamic scoping uses current environment

y:*5*, f:(y:*5*, λx.x+y), y:*6*, x:*7*; x ⇒ *7*

y:*5*, f:(y:*5*, λx.x+y), y:*6*, x:*7*; y ⇒ *6*

y:*5*, f:(y:*5*, λx.x+y), y:*6*; f ⇒ (y:*5*, λx.x+y)

y:*5*, f:(y:*5*, λx.x+y), y:*6*; 7 ⇒ *7*

y:*5*, f:(y:*5*, λx.x+y), y:*6*, x:*7*; x+y ⇒ *13*

y:*5*, f:(y:*5*, λx.x+y), y:*6*; f 7 ⇒ *13*

y:*5*, f:(y:*5*, λx.x+y); 6 ⇒ *6*

y:*5*, f:(y:*5*, λx.x+y); let y = 6 in f 7 ⇒ *13*

y:*5*; fun x → x+y ⇒ (y:*5*, λx.x+y)

•; 5 ⇒ *5* y:*5*; let f = fun x → x+y in … in f 7 ⇒ *13*

•; let y = 5 in let f = fun x → x+y in let y = 6 in f 7 ⇒ *13*