Assignment 4

The Denotational Semantics of Scope Resolution

Sample Answer

First, we annotate the program's different sections (which happen to be regions of scope):

```
program A;
begin var x;                      k1
      proc C;
      begin          k2
        x = 2
      end;
      proc B;
      begin var x;       k3
        x = 1;
        call  C
      end;
   x = 0;
   call B
end
```

The derivation is obtained by following the valuations functions as follows:

1. $M\llbracket \textbf{program } A;\ k_1 \rrbracket$
2. $= M\llbracket k_1 \rrbracket s_0\ e_0$
3. $= M\llbracket x{=}0; \textbf{call } B \rrbracket s_0\ (M\llbracket \textbf{proc } C;\ k_2\ \textbf{proc } B;\ k_3 \rrbracket e_0)$

We work on the environment first:

4. $M\llbracket \textbf{var } x;\ \textbf{proc } C;\ k_2\ \textbf{proc } B;\ k_3 \rrbracket e_0$
5. $= M\llbracket \textbf{proc } C;\ k_2\ \textbf{proc } B;\ k_3 \rrbracket (M\llbracket \textbf{var } x \rrbracket e_0)$

The variable declaration gives an environment

6. updateenv(e_0, $\llbracket x \rrbracket$, l_0), where l_0 is the unique location returned by next_locn(). So $e_1 = \{(x, l_0)\}$. The declaration of the procedures gives an environment:
7. $M\llbracket \textbf{proc } C;\ k_2\ \textbf{proc } B;\ k_3 \rrbracket e_1$
8. $= M\llbracket \textbf{proc } B;\ k_3 \rrbracket\ (M\llbracket \textbf{proc } C;\ k_2 \rrbracket e_1)$

The declaration of C gives e_2:

9. updateenv(e_1, ⟦C⟧, $\lambda s.M⟦k_2⟧s\ e_1$)

So $e_2 = \{(x, l_0), (C, \lambda s.M⟦k_2⟧s\ e_1)\}$. Back to step 8:

10. = M⟦**proc** B; k_3⟧ e_2
11. = updateenv(e_2, ⟦B⟧, $\lambda s.M⟦k_3⟧s\ e_2$)

Call this e_3, where e_3 has C mapped to the function that executes C's body with environment e_2, and B mapped to the function that executes B's body with environment e_1, and x is mapped to l_0. i.e. $e_3 = \{(x, l_0), (C, \lambda s.M⟦k_2⟧s\ e_1), (B, \lambda s.M⟦k_3⟧s\ e_2)\}$

Back to step 3:

12. = M⟦x=0;**call** B⟧s_0 e_3
13. = M⟦**call** B⟧ (M⟦x=0⟧s_0 e_3) e_3

The execution of x=0 is:

14. = M⟦x=0⟧s_0 e_3
15. = update(s_0, acccessenv(e_3, ⟦x⟧), M⟦0⟧)

Call this s_1, where s_1 maps l_0 to the value 0. i.e. $s_1 = \{(l_0, 0)\}$. Back to step 14:

16. = M⟦**call** B⟧s_1 e_3
17. = ((accessenv(e_3, ⟦B⟧) s_1)

Thus we are applying the function mapped to B in e_3 to the store s_1 in which l_0 is mapped to 0.

18. = M⟦k_3⟧s_1 e_2

Note that the environment is the one stored when the declaration environment was updated, i..e. it is the one in which x is mapped to l_0, and C is declared as well. We have not yet executed B's body, so its declaration of x is not yet in force.

19. = M⟦x=1; **call** C⟧s_1 (M⟦var x⟧e_2)

The new environment, call it e_4, has x mapped to l_1 instead of l_0. This is "shadowing" of a variable declared in an outer environment. i.e. $e_4 = \{(x, l_1), (C, \lambda s.M⟦k_2⟧s\ e_1)\}$. So:

20. = M⟦x=1; **call** C⟧s_1 e_4, where e_4 maps x to l_1
21. = M⟦**call** C⟧ (M⟦x=1⟧s_1 e_4) e_4

The execution of x=1 gives a store s_2

22. = update(s_1, accessenv(e_4, ⟦x⟧), M⟦1⟧)

i.e. l_1 is mapped to 1 in s_2. $s_2 = \{(l_0, 0), (l_1, 1)\}$ The call to C is then:

23. = M⟦**call** C⟧s_2 e_4
24. = ((accessenv(e_4, ⟦C⟧) s_2)
25. = M⟦k_2⟧s_2 e_1

Note again the C's environment (e_1) is the one stored with the function when it was declared. It only contains a mapping for x to l_0.

26. $= M[\![x=2]\!] s_2\ e_1$
27. $= update(s_2, accessenv(e_1, [\![x]\!]), M[\![2]\!])$
28. $= \{(l_0, 2), (l_1, 1)\}$

Since s_2 contains l_0 mapped to 0 and l_1 mapped to 1, and e_1 contains a mapping from x to l_0, it is l_0 that is updated to 2, not l_1. This is static scoping. Dynamic scoping can be obtained by storing a function of both store *and* environment when a procedure is declared, and applying this function, when the procedure is called, to the store and the environment at the point of call. This will change the value for l_1 instead of for l_0.