

Assignment 4

The Denotational Semantics of Scope Resolution

Goals

To understand the differences between static and dynamic scoping through denotational models of variable storage and statement execution.

Procedure

At the end of the assignment is a denotational definition of a highly simplified language that can have nested procedures, like Pascal. Read the definition thoroughly before starting the assignment. Variables and procedures are declared in an environment mapping. Variable identifiers map to a location (the function `next-locn` produces unique locations) and procedure identifiers map to a function that, when applied to a store executes the body of the procedure.

Consider the following program:

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

1. Write out the derivation for the program using the valuation functions and domain operations as given in the language definition. Use abbreviations wherever necessary to avoid writing too much. For example, call the whole program P, and then say:

$M[[P]]_{s_0} e_0$, rather than $M[[\text{program A}; \text{begin} \dots]]_{s_0} e_0$ etc.

2. What kind of scoping (static or dynamic) is exhibited by the semantic definitions as given? Explain your answer.
3. How would you change the definition to exhibit the other kind of scoping?

Hints

- Read the definition, then read it again.
- Make sure you understand what number of type of arguments are expected by each of the valuation and domain operation functions.
- Try to derive a very simple program first to see what happens. Something like **program P; begin var x; x = 1 end** for example will show you how to work the functions and produce a store with x's location mapped to 1. The derivation for this is:

$$\begin{aligned}
& M[\text{program P; begin var x; x = 2 end}] \\
&= M[\text{begin var x; x = 2}] s_0 e_0 \\
&= M[x = 2] s_0 (M[\text{var x}]e_0) \\
&= M[x = 2] s_0 \text{updateenv}(e_0, [x], l_0) \quad \text{where } l_0 \text{ is got from next-locn} \\
&= M[x = 2] s_0 e_1 \quad \text{where } e_1 = \{(x, l_0)\} \\
&= \text{update}(s_0, \text{accessenv}(e_0, [x]), M[2]) \\
&= \text{update}(s_0, l_0, 2) \\
&= \{(l_0, 2)\}
\end{aligned}$$

- which is the final store. (Notice how x is no longer around, just its location.)
- Find a good way to abbreviate the new stores and environments as they change so that each derivation step is clear (e_1 , e_2 , etc.)
 - Pay attention to which store and which environment is involved at each step.
 - Function calls are sometimes written as $f a b c$ and sometimes as $f(a,b,c)$. There is no difference intended.

Submission

Hard copy please. Typing your answers is good, but not essential if your handwriting is clear.

Grading

Total 50 points. Partial credit will be available for answers that are along the right lines.

Due Date

April 11th 5:00pm.

Language definition:

Abstract syntax:

$P \in \text{Program}$
 $K \in \text{Block}$
 $D \in \text{Declaration}$
 $S \in \text{Statement}$
 $I \in \text{Identifier}$
 $N \in \text{Numeral}$

$P ::= \text{program } I;K.$
 $K ::= \text{begin } D; S \text{ end}$
 $D ::= D_1;D_2 \mid \text{proc } I;K \mid \text{var } I \mid \epsilon$ [there could be no declarations]
 $S ::= S_1;S_2 \mid I=N \mid \text{call } I$

Semantic algebras:

I.-II. Natural numbers, identifiers
(not defined here, but assume the obvious meanings)

III. Expressible values [just natural numbers here]
Domain $\text{val} \in \text{Expressible-value} = \text{Nat}$

IV. Storage locations [the semantic equivalent of machine addresses]
Domain $l \in \text{Location}$
Operations
 $\text{next-locn} : \rightarrow \text{Location}$ [next-locn magically produces a new location every time it is called (no arguments)]

V. Denotable values [what is bound to declared identifiers in the environment]
Domain $d \in \text{Denotable-value} = \text{Location} + \text{Proc}$ [i.e. d can be either a Location or a Proc]

Where $\text{Proc} = \text{Store} \rightarrow \text{Store}$

VI. Environment [a map between identifiers and locations or procedures]
Domain $e \in \text{Environment} = \text{Id} \rightarrow \text{Denotable-value}$
Operations
 $\text{accessenv} : \text{Id} \rightarrow \text{Environment} \rightarrow \text{Denotable-value}$
 $\text{accessenv} = \lambda e. \lambda i. e(i)$
 $\text{updateenv} : \text{Environment} \rightarrow \text{Id} \rightarrow \text{Denotable-value} \rightarrow \text{Environment}$

updateenv = $\lambda e. \lambda i. \lambda l. [i \mapsto l]e$ [i.e. the environment e is changed so that the identifier i maps to the location l]

VII. Storable values [just natural numbers]

Domain $v \in \text{Storable-value} = \text{Nat}$

VIII. Stores [a map between locations and values]

Domain $s \in \text{Store} = \text{Location} \rightarrow \text{Storable-value}$

Operations

access: $\text{Location} \rightarrow \text{Store} \rightarrow \text{Storable-value}$

access = $\lambda l. \lambda s. s(l)$

update: $\text{Location} \rightarrow \text{Storable-value} \rightarrow \text{Store} \rightarrow \text{Store}$

update = $\lambda l. \lambda v. \lambda s. [l \mapsto v]s$ [i.e. the store s is changed so that the location l maps to the value v]

Valuation functions:

$M[\text{program } I; K] = M[K] s_0 e_0$ [e_0 is the empty environment, s_0 is the empty store]

$M[\text{begin } D; S \text{ end}] = \lambda s. \lambda e. M[S]s (M[D]e)$

$M[D_1; D_2] = \lambda e. M[D_2] (M[D_1]e)$

$M[N] = n$ [$n \in \text{Nat}$]

$M[\text{var } I] = \lambda e. \text{updateenv}(e, [I], \text{next-locn}())$

$M[\text{proc } I; K] = \lambda e. \text{updateenv}(e, [I], \lambda s. M[K]s e)$ [I is bound to a function in the environment]

$M[S_1; S_2] = \lambda s. \lambda e. M[S_2](M[S_1]s e)$

$M[I=N] = \lambda s. \lambda e. \text{update}(s, \text{accessenv}(e, [I]), M[N])$

$M[\text{call } I] = \lambda s. \lambda e. ((\text{accessenv}(e, [I]) s)$