Dynamic Scoping: A Lesson from Language Design Bugs

Variables and Mutation

**Imperative Computation**

\[
e ::= x \mid \ldots \mid x = e
\]

**Assignment**

\[
(x := e)
\]

\[
\downarrow (x := e) \Rightarrow e_1 \mid e_1 ; e_2 \mid \ldots
\]

\[
u_1; e_2 \Rightarrow e_2
\]

\[
((x := e_1) (v_2) \Rightarrow [v_2/x] e_1)
\]
Exercise: Come up with an expression (with \( \epsilon \)) that after some steps of eval becomes "non-sense." (original input expression "makes sense")

\[
( (x : \text{Int}) \Rightarrow \\
  x = 1 ; \\
  \text{print}(x) ) \ (2)
\]

\[
2 \leftarrow 1 ; \\
\text{print}(2)
\]

"non-sense" substitution doesn't work anymore

E \text{ environment} E : \text{Var} \rightarrow_{\text{fin}} \text{Vval}

Can't use substitution, so let's make explicit a "value environment" \( \wp \) like a memory store
\[ \langle E, e \rangle \rightarrow \langle E', e' \rangle \]

"Configuration" or "state" of my abstract machine

E will contain bindings for the free variables of e \( \langle E, e \rangle \)

Example Config. and "intuitive" steps

\[
\begin{align*}
\langle [x \mapsto 3], \frac{x = 4}{x} \rangle &\rightarrow \langle [x \mapsto 4], () ; x \rangle \\
&\rightarrow \langle [x \mapsto 4], x \rangle \\
&\rightarrow \langle [x \mapsto 4], 4 \rangle
\end{align*}
\]
This is dynamic scoping!

\[ n' = n_1 + n_2 \]

What is dynamic scoping?
The binding used depends on execution (i.e., the sequence of function calls)
\((x : \text{Int}) \Rightarrow ((x : \text{Int}) \Rightarrow 2)(3) + x)(1)\)

Dynamic scoping says 5!

Is the problem only related to variables not being "popped" on return?

No!

Example:

\[
\begin{align*}
\text{val } x & : \text{Int} = 1 \land \text{val } e_1 \land \text{val } e_2 \\
\text{val } g & : \text{Int} \Rightarrow \text{Int} = (y : \text{Int}) \Rightarrow x \land \text{val } h \land \text{val } h(3) = (x : \text{Int}) \Rightarrow g(z) \land \text{val } h(3)
\end{align*}
\]

What does this return under dynamic scoping?

\[
\begin{align*}
\rightarrow [x \mapsto 1], \text{ val } g : \text{Int} \land \text{val } h \\
\rightarrow [x \mapsto 1, g \mapsto (y : \text{Int}) \Rightarrow x], \text{ val } h...
\end{align*}
\]
Still have dynamic scoping

This is the "feature" introduced by Lisp — Language design is subtle

Can fix explicit environments

— need closures

We are instead going to use substitution

+ "memory addresses"
\[ e ::= \ldots | a | \ast a = e_1 | \ast a \]

\[ V ::= \ldots | a \]

memory \( M : \text{Addr} \rightarrow \text{Val} \)

\[ \langle M, e \rangle \rightarrow \langle M', e' \rangle \]

\[ \langle M, \ast a \rangle \rightarrow \langle M', M(a) \rangle \]

\[ \langle M, \ast a := v \rangle \rightarrow \langle M[a \rightarrow v], () \rangle \]
\[ a \not\in \text{dom}(M) \]

\[ \langle M_1((x:2) \Rightarrow e_1)(v_2) \rangle \rightarrow \]

\[ \langle M[a \rightarrow v_2], [x/a]e_1 \rangle \]

\[ \uparrow \]

\[ a \text{ is "fresh" or "new"} \]