Variables, Type Structures, and Modules

Programming Languages CS442

David Toman

School of Computer Science
University of Waterloo

Variable Declarations

We want $D ::= \ldots \mid \mathbf{var}\, V$

$\Rightarrow$ the declaration has two functions:

1. allocate *fresh* location:

$$allocate(\langle n_1, n_2, \ldots, n_k \rangle) = (loc_{k+1}, \langle n_1, n_2, \ldots, n_k, 0 \rangle)$$

2. bind the identifier I to the new location:

$$[\![\pi \vdash \mathbf{var}\, V : \{V : intloc\} dec]\!] e\ s = (\{V = I\}, s') \quad where\ \ (I, s') = allocate\ s$$

Note

Declarations now modify the store! *Impact on the glue?/Sequencing?*

Improved Declarations

$$[\![\pi \vdash D : \pi_1 dec]\!] : [\![\pi]\!] \to Store_\perp \to [\![\pi_1]\!] \times Store_\perp$$

Other Abstractions:

$$[\![\pi \vdash \textbf{define}\, I = U : \{I : \theta\} dec]\!] e\ s = \{I = f\}, s$$
$$\text{where } f = [\![\pi \vdash U : \theta]\!] e$$

Glue: $[\![\emptyset \vdash D\ \textbf{in}\ C : comm]\!] s = [\![\pi \vdash C : comm]\!] e\ s'$

$$\text{where } e, s' = [\![\emptyset \vdash D : \pi dec]\!] \emptyset\ s$$

$$[\![\pi \vdash D_1, D_2 : (\pi_1 \uplus \pi_2) dec]\!] e\ s = e_1 \cup e_2, s''$$
$$\text{where } e_1, s' = [\![\pi \vdash D_1 : \pi_1 dec]\!] e\ s$$
$$\text{and } e_2, s'' = [\![\pi \vdash D_1 : \pi_1 dec]\!] e\ s'$$

Variables vs. Type Structures

Not quite satisfactory. . .

Idea

Separate allocation of storage from identifier binding.

Type Structures (= storage allocation *code*)

$$
\begin{aligned}
T &::= \textbf{newint} \mid I \mid \textbf{record}\, D\, \textbf{end} \\
D &::= \ldots \mid \textbf{var}\, V : T \mid \textbf{class}\, I = T \\
X &::= I \mid X.I
\end{aligned}
$$

+ names for type structures
+ compound type structures (needs records)

Type Structures: Typing Rules

Type Structures:

$$\pi \vdash \textbf{newint} : \mathit{intloc\ class}$$

$$\frac{\pi \vdash D : \pi_1 dec}{\pi \vdash \textbf{record}\, D\, \textbf{end} : \pi_1 class}$$

Declarations:

$$\frac{\pi \vdash T : \delta class}{\pi \vdash \textbf{var}\, V : T : \{V : \delta\} dec}$$

$$\frac{\pi \vdash T : \delta class}{\pi \vdash \textbf{class}\, C = T : \{C : \delta class\} dec}$$

Identifier Expressions:

$$\pi \vdash I : \theta \ \ \mathit{if}\ \ (I : \theta) \in \pi$$

$$\frac{\pi \vdash X : \pi_1}{\pi \vdash X.I : \theta}\ \ \mathit{if}\ \ (I : \theta) \in \pi_1$$

Type Structures: Semantics

Type Structures:

$[\![\pi \vdash \textbf{newint} : \mathit{intloc\ class}]\!]e\ s = \mathit{allocate}(s)$

$[\![\pi \vdash \textbf{record}\ D\ \textbf{end} : \pi \mathit{class}]\!]e\ s = [\![\pi \vdash D : \pi \mathit{dec}]\!]e\ s$

Declarations:

$[\![\pi \vdash \textbf{define}\ I = U : \{I : \theta\}\mathit{dec}]\!]e\ s = \{I = f\}, s \quad f = [\![\pi \vdash U : \theta]\!]e$

$[\![\pi \vdash \textbf{var}\ I : T : \{I : \delta\}\mathit{dec}]\!]e\ s = \{I = v\}, s'$

$v, s' = [\![\pi \vdash T : \delta \mathit{class}]\!]e\ s$

$[\![\pi \vdash D_1, D_2 : (\pi_1 \cup \pi_2)\mathit{dec}]\!]e\ s = e_1 \cup e_2, s''$

$e_1, s' = [\![\pi \vdash D_1 : \pi_1 \mathit{dec}]\!]e\ s, \quad e_2, s'' = [\![\pi \vdash D_1 : \pi_1 \mathit{dec}]\!]e\ s'$

Identifier Expressions:

$[\![\pi \vdash I : \theta]\!]e\ s = v\ \mathit{for}\ (I = v) \in e$

$[\![\pi \vdash X.I : \theta]\!]e\ s = v\ \mathit{for}\ (I = v) \in [\![\pi \vdash X : \pi_1]\!]e$

Abstractions of Declarations

Syntax:

$$D ::= \ldots \mid \textbf{module}\, M = \{D\} \mid \textbf{import}\, M$$

Typing rules:

$$\frac{\pi \vdash D : \pi_1 dec}{\pi \vdash \textbf{module}\, M = \{D\} : \{M : \pi_1 dec\} dec}$$

$$\pi \vdash \textbf{import}\, M : \pi_1 dec \;\; \textit{if} \;\; (M : \pi_1 dec \in \pi$$

Semantics for Modules

Idea

Variables declared in a module should be allocated just once

$\Rightarrow$ ***import****ing a module allows "sharing"*

$$[\![\pi \vdash \textbf{module}\, M = \{D\} : \{M : \pi_1 dec\} dec]\!] e\ s = (\{M = e_1\}, s_1)$$
$$\textit{where}\ \ (e_1, s_1) = [\![\pi \vdash D : \pi_1 dec]\!] e\ s$$

$$[\![\pi \vdash \textbf{import}\, M : \pi_1 dec]\!] e\ s = (e_1, s) \textit{for}\ \ (e_1, s) \in e$$

Summary

- Variables are NOT JUST bindings

  $\Rightarrow$ unlike most other “declarations”

  $\Rightarrow$ variable declarations are “commands”

  that return values in addition to $Store_{\perp}$

- **class**es and **module**s give names to declarations

  $\Rightarrow$ but they differ in their evaluation mode (lazy/eager)

- Questions:
  1. can we have “initialization” expressions (**var** A : **newint** $:= E$)?
  2. how would a **return E** “command” look like?
  3. can we **import** modules multiple times (e.g., via other modules)?