OpSem Theory

COMP105 Fall 2015

James McCants

Problem 16

Here are the standard ImpCore inferences rules for `VAR(x)`:

$$\frac{x \in \mathsf{dom}\,\rho}{\langle \mathsf{VAR}(x),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle \rho(x),\ \xi,\ \phi,\ \rho\rangle}$$

and

$$\frac{x \notin \mathsf{dom}\ \rho \qquad x \in \mathsf{dom}\,\xi}{\langle \mathsf{VAR}(x),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle \xi(x),\ \xi,\ \phi,\ \rho\rangle}$$

(a) Awk-like semantics

Add 'VAR(x)':

$$\frac{x \notin \mathsf{dom}\ \rho \qquad x \notin \mathsf{dom}\,\xi}{\langle \mathsf{VAR}(x),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle 0,\ \xi'(x \to 0),\ \phi,\ \rho\rangle}$$

Add 'SET(x)':

$$\frac{x \notin \mathsf{dom}\ \rho\, x \notin \mathsf{dom}\,\xi \qquad \langle e,\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle e,\ \xi',\ \phi,\ \rho'\rangle}{\langle \mathsf{SET}(x,\ e),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle v,\ \xi'(x \to v),\ \phi,\ \rho'\rangle}$$

(b) Icon-like semantics

Add 'VAR(x)':

$$\frac{x \notin \mathsf{dom}\ \rho \qquad x \notin \mathsf{dom}\,\xi}{\langle \mathsf{VAR}(x),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle 0,\ \xi,\ \phi,\ \rho'(x \to 0)\rangle}$$

Add 'SET(x)':

$$\frac{x \notin \mathsf{dom}\ \rho\, x \notin \mathsf{dom}\,\xi \qquad \langle e,\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle e,\ \xi,\ \phi,\ \rho'\rangle}{\langle \mathsf{SET}(x,\ e),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle v,\ \xi,\ \phi,\ \rho'(x \to v)\rangle}$$

(c) Which do you prefer and why?

I prefer the change to Icon because keeping variables that can be declared implicitly in a local environment seems safer. It limits the possibility to break things that rely on the global environment.

Problem 13

$$\frac{\dfrac{\dfrac{x \in \mathsf{dom}\,\rho \qquad \rho(x)\ =\ 99}{\langle \mathsf{VAR}(x),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle 99,\ \xi,\ \phi,\ \rho\rangle} \quad \dfrac{}{\langle \mathsf{LITERAL}(3),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle 3,\ \xi,\ \phi,\ \rho\rangle}}{\langle \mathsf{SET}(\mathsf{VAR}(x),\ \mathsf{LITERAL}(3)),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle 3,\ \xi',\ \phi,\ \rho'(x \to 3)\rangle} \quad \dfrac{x \in \mathsf{dom}\,\rho' \qquad \rho'(x)\ =\ 3}{\langle \mathsf{VAR}(x),\ \xi',\ \phi,\ \rho'\rangle \Downarrow \langle 3,\ \xi',\ \phi,}}{\langle \mathsf{BEGIN}((\mathsf{SET},\ \mathsf{VAR}(x),\ \mathsf{LITERAL}(3))\ \mathsf{VAR}(x)),\ \xi,\ \phi,\ \rho\rangle \Downarrow \langle 3,\ \xi',\ \phi,\ \rho'\rangle}$$

The cut off line:

$\langle \mathsf{VAR}(x),\, \xi',\, \phi,\, \rho' \rangle \Downarrow \langle 3,\, \xi',\, \phi,\, \rho' \rangle$

Problem 14

IfTrue:

$$\frac{\langle \mathsf{VAR}(x),\, \xi,\, \phi,\, \rho \rangle \Downarrow \langle v_1,\, \xi,\, \phi,\, \rho \rangle \qquad v_1 \neq 0 \qquad \langle \mathsf{VAR}(x),\, \xi,\, \phi,\, \rho \rangle \Downarrow \langle v_2,\, \xi'',\, \phi,\, \rho'' \rangle}{\langle \mathsf{IF}(\mathsf{VAR}(x),\, \mathsf{VAR}(x),\, \mathsf{LITERAL}(0)),\, \xi,\, \phi,\, \rho \rangle \Downarrow \langle v_2,\, \xi'',\, \phi,\, \rho'' \rangle}$$

In this case: $v_1 = v_2 \neq 0$

IfFalse:

$$\frac{\langle \mathsf{VAR}(x),\, \xi,\, \phi,\, \rho \rangle \Downarrow \langle v_1,\, \xi,\, \phi,\, \rho \rangle \qquad v_1 = 0 \qquad \langle \mathsf{LITERAL}(0),\, \xi,\, \phi,\, \rho \rangle \Downarrow \langle v_2,\, \xi'',\, \phi,\, \rho'' \rangle}{\langle \mathsf{IF}(\mathsf{VAR}(x),\, \mathsf{VAR}(x),\, \mathsf{LITERAL}(0)),\, \xi,\, \phi,\, \rho \rangle \Downarrow \langle v_2,\, \xi'',\, \phi,\, \rho'' \rangle}$$

In this case: $v_1 = v_2 = 0$

Problem 23

Base Cases

Literal: a) In the case of Literal ρ is popped off the stack and pushed back on with no change. b) Because ρ is not changed in any way nothing is thrown away. There is no possibility for the stack to be missing environments.

FormalVar: a) In the case of FormalVar ρ is popped off the stack, checked for x, and then pushed back on when x is found. b) Because ρ is not changed in any way nothing is thrown away. There is no possibility for the stack to be missing environments .

GlobalVar: a) In the case of GlobalVar ρ is popped off the stack, checked for x, and then pushed back on when x is not found. b) Because ρ is not changed in any way nothing is thrown away. There is no possibility for the stack to be missing environments.

EmptyBegin: a) In the case of EmptyBegin ρ is popped off the stack and pushed back on with no change. b) Because ρ is not changed in any way nothing is thrown away. There is no possibility for the stack to be missing environments.

ApplyAdd: a) In the case of ApplyAdd ρ is not used in the addition so it can stay on the stack. b) Because ρ is not used it is not changed and nothing is thrown away . There is no possibility for the stack to be missing environments.

Induction Steps

FormalAssign: a) In the case of FormalAssign ρ is popped off the stack and a recursive call to eval is made to find what x should be set to. Once x is modified to its new value ρ', which contains the updated x, is pushed onto the stack and the old ρ is thrown away. b) No environments have been lost on the stack in this procedure because $\rho\prime$ contains the all of ρ with just x updated.

IfTrue: a) In the case of IfTrue ρ is popped off the stack and a call to eval is made to determine the case of the if statement. Any change to ρ results in the environment being copied with the change recorded. The updated environment is $\rho\prime$ and then a second call to eval is made to check what to do when true. Any changes here are recorded similarily in $\rho\prime\prime$ which is then pushed on the stack. b) At every step something changes ρ is copied with the change recorded into some $\rho\prime$ and then ρ is thrown out. There is no loss of environment on the stack.

IfFalse: a) In the case of IfFalse ρ is popped off the stack and a call to eval is made to determine the case of the if statement. Any change to ρ results in the environment being copied with the change recorded. The updated environment is $\rho\prime$ and then a second call to eval is made to check what to do when false. Any changes here are recorded similarily in $\rho\prime\prime$ which is then pushed on the stack. b) At every step something changes ρ is copied with the change recorded into some $\rho\prime$ and then ρ is thrown out. There is no loss of environment on the stack.

WhileIterate: a) In the case of WhileIterate ρ is popped off the stack and eval is called on e_1, any change is recorded and creates $\rho\prime$. If $v_1 \neq 0$ then e_2 is evaluated and any change to $\rho\prime$ is recorded in $\rho\prime\prime$. The whole thing then recursively calls itself starting with $\rho\prime\prime$ as the initial environment. b) $\rho\prime$ is not pushed back on the stack until WhileEnd so this is addressed there.

WhileEnd: a) In the case of WhileEnd ρ is popped off the stack and eval is called on e_1, any change is recorded and creates $\rho\prime$. If $v_1 = 0$ then $\rho\prime$ is pushed onto the stack. b) At every step something changes ρ is copied with the change recorded into some $\rho\prime$ and then ρ is thrown out. There is no loss of environment on the stack.

Begin: a) In the case of Begin, ρ is popped off of the stack and then every e_n is evaluated. Each evaluation results in ρ_n being changed to $\rho_n\prime$. Where the old environment is copied with the changes recorded. When the last e_n has been evaluated then $\rho\prime$ is pushed onto the stack. b) At every step something changes ρ is copied with the change recorded into some $\rho\prime$ and then ρ is thrown out. There is no loss of environment on the stack.

ApplyUser: a) In the case of ApplyUser, ρ is popped off of the stack and then every e_n is evaluated. Each evaluation results in ρ_n being changed to $\rho_n\prime$. Where the old environment is copied with the changes recorded. When the last e_n has been evaluated then $\rho\prime$ is pushed onto the stack. b) At every step something changes ρ is copied with the change recorded into some $\rho\prime$ and then ρ is thrown out. There is no loss of environment on the stack.