Featherweight Breeze: Step 2/4

Cătălin Hriţcu, Benoît Montagu, Benjamin C. Pierce, and the Breeze team

December 14, 2011

1 Syntax

Untyped lambda calculus with booleans, pairs, classification, and first-class labels.

$L,\ H,\ pc$	$::=$			label
	$\vert$	$\top$	M	top secret
	$\vert$	$\bot$	M	unclassified
	$\vert$	$L_1 \vee L_2$	M	label join
	$\vert$	(L)	S	
c	$::=$			constants
	$\vert$	$()$		unit
	$\vert$	true		true
	$\vert$	false		false
	$\vert$	L		label
t	$::=$			terms
	$\vert$	c		constant
	$\vert$	x		variable
	$\vert$	$\lambda x.t$	$\mathsf{bind}\ x\ \mathsf{in}\ t$	abstraction
	$\vert$	$t_1\ t_2$		application
	$\vert$	$\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2$	$\mathsf{bind}\ x\ \mathsf{in}\ t_2$	let
	$\vert$	(t_1, t_2)		pairing
	$\vert$	$\mathsf{fst}\ t$		first projection
	$\vert$	$\mathsf{snd}\ t$		second projection
	$\vert$	$\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3$		conditional
	$\vert$	$t_1 == t_2$		equality on constants
	$\vert$	$t_1 @ t_2$		classify t_1 with label t_2
	$\vert$	$\mathsf{labelOf}\ t$		returns the label of t
	$\vert$	(t)	S	

v	$::=$			values
	$\mid$	c		constants
	$\mid$	$\langle \rho,\ \lambda x.\, t \rangle$	bind x in t	closures
	$\mid$	(a_1, a_2)		pairs
a	$::=$			atoms
	$\mid$	$v@L$		labeled value
ρ	$::=$			environments
	$\mid$	empty		
	$\mid$	$\rho, x : a$		
	$\mid$	(ρ)	S	

2 Evaluation with Dynamic IF Control

$\boxed{\rho, pc \vdash t \Downarrow a}$

$$\frac{}{\rho, pc \vdash c \Downarrow c@pc} \quad \textsc{Eval\_Const}$$

$$\frac{\rho(x) = v@L}{\rho, pc \vdash x \Downarrow v@(L \vee pc)} \quad \textsc{Eval\_Var}$$

$$\frac{}{\rho, pc \vdash (\lambda x.t) \Downarrow \langle \rho,\ \lambda x.\, t \rangle@pc} \quad \textsc{Eval\_Abs}$$

$$\frac{\begin{array}{l} \rho, pc \vdash t' \Downarrow \langle \rho',\ \lambda x.\, t \rangle@L' \\ \rho, pc \vdash t'' \Downarrow a'' \\ (\rho', x : a''), (pc \vee L') \vdash t \Downarrow a \end{array}}{\rho, pc \vdash t'\, t'' \Downarrow a} \quad \textsc{Eval\_App}$$

$$\frac{\begin{array}{l} \rho, pc \vdash t \Downarrow a \\ (\rho, x : a), pc \vdash t' \Downarrow a' \end{array}}{\rho, pc \vdash \mathsf{let}\, x = t\, \mathsf{in}\, t' \Downarrow a'} \quad \textsc{Eval\_Let}$$

$$\frac{\begin{array}{l} \rho, pc \vdash t' \Downarrow a' \\ \rho, pc \vdash t'' \Downarrow a'' \end{array}}{\rho, pc \vdash (t', t'') \Downarrow (a', a'')@pc} \quad \textsc{Eval\_Pair}$$

$$\frac{\rho, pc \vdash t \Downarrow (v'@L', a'')@L}{\rho, pc \vdash \mathsf{fst}\, t \Downarrow v'@(L' \vee L)} \quad \textsc{Eval\_Fst}$$

$$\frac{\rho, pc \vdash t \Downarrow (a', v''@L'')@L}{\rho, pc \vdash \mathsf{snd}\, t \Downarrow v''@(L'' \vee L)} \quad \textsc{Eval\_Snd}$$

$$\frac{\begin{array}{l} \rho, pc \vdash t \Downarrow \mathsf{true}@L \\ \rho, (pc \vee L) \vdash t' \Downarrow a' \end{array}}{\rho, pc \vdash \mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' \Downarrow a'} \quad \textsc{Eval\_If\_True}$$

$$\frac{\begin{array}{c}\rho, pc \vdash t \Downarrow \mathsf{false}@L \\ \rho, (pc \vee L) \vdash t'' \Downarrow a''\end{array}}{\rho, pc \vdash \mathsf{if}\ t\ \mathsf{then}\ t'\ \mathsf{else}\ t'' \Downarrow a''}\ \textsc{Eval\_If\_False}$$

$$\frac{\begin{array}{c}\rho, pc \vdash t' \Downarrow c'@L' \\ \rho, pc \vdash t'' \Downarrow c''@L'' \\ v \triangleq c' = c''\end{array}}{\rho, pc \vdash t' == t'' \Downarrow v@(L' \vee L'')}\ \textsc{Eval\_Eq}$$

$$\frac{\begin{array}{c}\rho, pc \vdash t \Downarrow v@L \\ \rho, pc \vdash t' \Downarrow L'@L''\end{array}}{\rho, pc \vdash t@t' \Downarrow v@(L \vee L' \vee L'')}\ \textsc{Eval\_Classify}$$

$$\frac{\rho, pc \vdash t \Downarrow v@L}{\rho, pc \vdash \mathsf{labelOf}\ t \Downarrow L@L}\ \textsc{Eval\_LabelOf}$$

3 Changes wrt Step 1

- Added labels as abstract constants. The only operation we assume on labels is join: $L_1 \vee L_2$, computing the least secret/tainted label that is more secret/tainted than both L_1 and L_2.
- Added new classification construct $t_1@t_2$
- Evaluation produces an atom = value together with its label (environments also store atoms now). Breeze does fine-grained dynamic information flow control (IFC), so all values are labeled.
- Added *pc* label to the semantics in order to track implicit flows. The *pc* is the least upper bound of the labels of all values on which the program has currently branched. In this variant of Featherweight Breeze, the *pc* is automatically lowered on control flow merge points (e.g. the end of a conditional). Rule Eval_Let shows that the *pc* is not threaded through sequentially, and is automatically restored on control-flow merge points. The *pc* infects all resulting values in order to preserve soundness in the presence of automatic *pc* lowering/restoring.
- Added "castrated" labelOf construct. It is sound but useless, and shows that labels can't be made public in this setting, leading to the "poison pill" problem.
- Added equality on constants; useful in (counter)examples, since it also works on labels.

4 Counterexamples

- We need "infectious *pc*" to prevent implicit flows in the presence of automatic *pc* lowering/restoring:
  $\mathsf{let}\ copy = (\mathsf{if}\ x@H\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false})\ \mathsf{in}\ publish\ copy$

- Exercise: Rule EVAL_APP needs to raise the pc; encode Church booleans to see that.
- Labels are an information-flow channel, we can't have unrestricted labelOf in the presence of automatic pc lowering/restoring:
  $\mathsf{let}\ y = (\mathsf{if}\ x@H\ \mathsf{then}\ ()@H\ \mathsf{else}\ ()@\top)\ \mathsf{in}\ \mathit{publish}\ ((\mathsf{labelOf}\ y) == H)$.

5 Termination-insensitive Non-interference

Claim 1 (Infectious PC). *If $\rho, pc \vdash t \Downarrow v@L$ then $pc \sqsubseteq L$.*

Claim 2 (Non-interference). *If $\rho_1, pc \vdash t \Downarrow a_1$, and $\rho_2, pc \vdash t \Downarrow a_2$, and $\rho_1 \simeq_l \rho_2$, then $a_1 \simeq_L a_2$.*
(For some "unspecified" definitions of $\rho_1 \simeq_l \rho_2$ and $a_1 \simeq_L a_2$)

6 Two Problems (fixed in Step 3)

6.1 The "Infectious *pc*" Problem

$\mathit{empty}, \bot \vdash \mathsf{if}\,(\mathsf{true}@H)\,\mathsf{then}\,(\mathsf{true}, (\mathsf{false}, ()))\,\mathsf{else}\,() \Downarrow (\mathsf{true}@H, (\mathsf{false}@H, ()@H)@H)@H$

6.2 The "Poison Pill" Problem

- Breeze does fine-grained, dynamic IFC with decentralized LM
- Any code can classify data:

```
let (P,_,__) = newPrin "P" in
let pill = 42@(P -> [P]) in
```

- High data can be hidden under low labels

```
let xs = [1,2,pill]@L in
send cpub x
```

- IFC violations are dynamic errors

```
let xs = recv cpub;
let y = find ((==) 3) xs;
send cpub' y
```

- If threads get killed on IFC errors (like in actual Breeze) critical system components get killed on reading poison pill. Even without access control checks the pc infects the result y, which then can make other things fail, like the `send cpub' y` at the end.