1 Syntax

\[ L, H, pc \ ::= \begin{array}{ll}
\text{label} & \\
\top & M \quad \text{top secret} \\
\bot & M \quad \text{unclassified} \\
L_1 \lor L_2 & M \quad \text{label join} \\
(L) & S
\end{array} \]

\[ c \ ::= \begin{array}{ll}
() & \text{unit} \\
\text{true} & \text{true} \\
\text{false} & \text{false} \\
L & \text{label}
\end{array} \]

\[ t \ ::= \begin{array}{ll}
c & \text{constant} \\
x & \text{variable} \\
\lambda x.t & \text{abstraction} \\
t_1 \ t_2 & \text{application} \\
\text{let} \ x = t_1 \ \text{in} \ t_2 & \text{let} \\
(t_1, t_2) & \text{pairing} \\
\text{fst} \ t & \text{first projection} \\
\text{snd} \ t & \text{second projection} \\
\text{if} \ t_1 \ \text{then} \ t_2 \ \text{else} \ t_3 & \text{conditional} \\
t_1 == t_2 & \text{equality on constants} \\
t_1 @ t_2 & \text{classify} \ t_1 \ \text{with label} \ t_2 \\
t_1 \langle t_2 \rangle & \text{executes} \ t_2, \ \text{labels result with} \ t_1, \ \text{restores} \ pc \\
\text{labelOf} \ t & \text{returns the label of} \ t \\
\text{getPc} () & \text{returns the current pc} \\
\text{valueOf} \ t & \text{takes label of} \ t \ \text{and joins it to} \ pc
\end{array} \]
wrong attempt to define brackets uses this

<table>
<thead>
<tr>
<th>[t]</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t)</td>
<td>S</td>
</tr>
</tbody>
</table>

\[
v ::= \begin{cases} \text{values} \\ c \quad \text{constants} \\ (\rho, \lambda x. t) \quad \text{bind } x \text{ in } t \\ (a_1, a_2) \quad \text{closures} \end{cases} \\
\]

\[
a ::= \begin{cases} \text{atoms} \\ v@L \quad \text{labeled value} \end{cases} \\
\]

\[
\rho ::= \begin{cases} \text{environments} \\ \text{empty} \\ \rho, x : a \\ (\rho) \quad \text{S} \end{cases} \\
\]

## 2 Evaluation with Dynamic IF Control

\[
\rho \vdash t, pc \Downarrow a, pc' \\
\]

\[
\begin{array}{l}
\rho \vdash c, pc \Downarrow ca\bot, pc \\
\hline
\text{Eval}_\text{CONST} \\
\rho(x) = a \quad \rho \vdash x, pc \Downarrow a, pc \\
\hline
\text{Eval}_\text{VAR} \\
\rho \vdash (\lambda x. t), pc \Downarrow (\rho, \lambda x. t)@\bot, pc \\
\rho \vdash t', pc \Downarrow (\rho', \lambda x. t)@L', pc' \\
\rho \vdash t'', pc' \Downarrow a'', pc'' \\
(\rho', x : a'') \vdash t, (pc'' \lor L') \Downarrow a, pc''' \\
\rho \vdash (t' t''), pc \Downarrow a, pc''' \\
\hline
\text{Eval}_\text{ABS} \\
\rho \vdash t', pc \Downarrow a', pc' \\
(\rho, x : a') \vdash t'', pc' \Downarrow a'', pc'' \\
\rho \vdash (\text{let } x = t' \text{ in } t''), pc \Downarrow a'', pc'' \\
\hline
\text{Eval}_\text{APP} \\
\rho \vdash t', pc \Downarrow a', pc' \\
\rho \vdash t'', pc' \Downarrow a'', pc'' \\
\rho \vdash (t', t''), pc \Downarrow (a', a'')@\bot, pc'' \\
\hline
\text{Eval}_\text{LET} \\
\rho \vdash t, pc \Downarrow (v@L', a'')@L, pc' \\
\rho \vdash (\text{fst } t), pc \Downarrow v@L', (pc' \lor L) \\
\hline
\text{Eval}_\text{FST} \\
\rho \vdash t, pc \Downarrow (a', v''@L''@L, pc' \\
\rho \vdash (\text{snd } t), pc \Downarrow v''@L'', (pc' \lor L) \\
\hline
\text{Eval}_\text{SND} \\
\end{array}
\]

2
\[
\begin{array}{ll}
\rho \vdash t, \text{pc} \downarrow \text{true} @ L, \text{pc}' & \text{Eval_IF_TRUE} \\
\rho \vdash t', (\text{pc}' \lor L) \downarrow a', \text{pc}'' & \\
\rho \vdash (\text{if } t \text{ then } t' \text{ else } t''), \text{pc} \downarrow a', \text{pc}'' & \\
\rho \vdash t, \text{pc} \downarrow \text{false} @ L, \text{pc}' & \text{Eval_IF_FALSE} \\
\rho \vdash t', (\text{pc}' \lor L) \downarrow a'', \text{pc}'' & \\
\rho \vdash t', \text{pc} \downarrow c' @ L', \text{pc}' & \\
\rho \vdash t'', \text{pc} \downarrow c'' @ L'', \text{pc}'' & \\
v \triangleq c' = c'' & \\
\rho \vdash (t' == t''), \text{pc} \downarrow v \oplus (L' \lor L''), \text{pc}'' & \text{Eval_EQ} \\
\rho \vdash t, \text{pc} \downarrow v @ L, \text{pc}' & \\
\rho \vdash t', \text{pc} \downarrow v @ L', \text{pc}' & \\
\rho \vdash t', \text{pc} \downarrow v @ L'', \text{pc}'' & \\
\rho \vdash (t @ t'), \text{pc} \downarrow v \oplus (L' \lor L''), (\text{pc}'' \lor L'') & \text{Eval_BRACKET} \\
\rho \vdash t', \text{pc} \downarrow L @ L', \text{pc}' & \\
\rho \vdash t'', (\text{pc}' \lor L') \downarrow v \oplus L'', \text{pc}'' & \\
L'' \lor \text{pc}'' \subseteq L \lor (\text{pc}' \lor L') & \\
\rho \vdash t'(t''), \text{pc} \downarrow v \oplus L, (\text{pc}' \lor L') & \\
\rho \vdash \text{labelOf } t, \text{pc} \downarrow L @ \bot, \text{pc}' & \text{Eval_LABELOF} \\
\rho \vdash \text{getPc } (), \text{pc} \downarrow \bot @ \bot, \text{pc} & \text{Eval_GETPC} \\
\rho \vdash t, \text{pc} \downarrow v @ L, \text{pc}' & \\
\rho \vdash \text{valueOf } t, \text{pc} \downarrow v @ (\text{pc}' \lor L) & \text{Eval_VALUEOF} \\
\end{array}
\]

3 Brackets

Brackets are constructs for executing a computation and restoring the initial pc when the computation ends.

3.1 First try

\[
\begin{array}{ll}
\rho \vdash t, \text{pc} \downarrow v @ L, \text{pc}' & \\
\rho \vdash [t], \text{pc} \downarrow v \oplus (L \lor \text{pc}'), \text{pc} & \\
\end{array}
\]

The main idea is to move some of the protection from the pc to the label on the resulting value. Since \(v\) is protected in the premise by \(L\) and by \(pc'\), in the result we can move all this protection to the label of \(v\), which is now \(L \lor \text{pc}'\), and the \(pc\) can safely be restored to the original one. The reason this doesn’t quite work is that labels are public, and while in the premise the label \(L\) is protected by \(pc'\), in the conclusion \(L\) would only be protected by the (potentially lower) \(pc\). Here is a counterexample exploiting the label channel:

\[
\text{let } y = \text{if } x @ H \text{ then } () @ H \text{ else } () @ \top \text{ in publish (labelOf } y) == H
\]
Another problem is that in the premise $pc'$ is protected by itself, while in the conclusion $pc'$ is protected only by $pc$. The counterexample for this looks as follows:

```plaintext
let y = [if x@H then raisePc H else raisePc ⊤] in publish (labelOf y) == H
where raisePc ≜ λx.((λy,y)@x) () (in step 4/4 we’ll add a raisePc primitive).
```

3.2 Second try: closing the label channel

\[
\begin{align*}
\rho ⊢ t, pc \Downarrow v@L'', pc'' \land pc'' \sqsubseteq L \land pc \\
\rho ⊢ L(t), pc \Downarrow v@L, pc
\end{align*}
\]

We close the label channel by requiring the user to choose in advance the label on the result. This way the label on the result cannot depend on secrets. This works but is still too restrictive: because of the $L'' \land pc'' \sqsubseteq L$ condition in the premise we cannot use brackets to classify values to a low label in a high context. For instance if $x@⊤$ then $⊥⟨true⟩$ else () fails, although if $x@⊤$ then true@⊥ else () works fine.

3.3 Third try: making brackets the ultimate classification construct

\[
\begin{align*}
\rho ⊢ t, pc \Downarrow v@L'', pc'' \land pc'' \sqsubseteq L \land pc \\
\rho ⊢ L(t), pc \Downarrow v@L, pc
\end{align*}
\]

The intuition is that the final value is not only protected by $L$, but also by the $pc$, so we can relax the premise of the rule from $L'' \land pc'' \sqsubseteq L \land pc$ to $L'' \land pc'' \sqsubseteq L \land pc$. This works but is still too restrictive, since the label $L$ is required to be a constant.

3.4 Fourth try: first-class labels on brackets

\[
\begin{align*}
\rho ⊢ t', pc \Downarrow L@⊥, pc' \\
\rho ⊢ t'', pc' \Downarrow v@L'', pc'' \land pc'' \sqsubseteq L \land pc \\
\rho ⊢ t'(t''), pc \Downarrow v@L, pc
\end{align*}
\]

This step is very easy, but only as long as the label on $L$ is required to be $⊥$.

3.5 Fifth try: the final rule

The final Eval_Bracket rule additionally takes care of the label on $L$ by raising the $pc$ appropriately, otherwise it’s the same as before.

4 Other Changes wrt Step 2

- Dropped automatic $pc$ lowering/restoring. Now threading the $pc$ through as a piece of state.
- Made $pc$ be non-infectious.
- Made all labels public: labelOf no longer protects the resulting label with itself, and added a new getPc construct.
• The old rules for pair projection (Eval\_Fst and Eval\_Snd) are unsound in our new public label setting. The fix is to join the outer pair label to the resulting pc.

• The old rule for classification would also be unsound in our new public label setting; fixed. Classification is anyway completely subsumed by brackets: \( t_1 \odot t_2 \triangleq \text{let } x = t_2 \text{ in } x\langle t_1 \rangle. \)

• Added new value\_Of construct that strips off the label of an atom and joins it to the pc. This is roughly the dual of brackets, which take taint from the pc and put it in the label of the result.

5 Counterexamples

• Here is why Eval\_Fst and Eval\_Snd had to change:
  \[
  \text{let } y = H\langle \text{if } x \odot H \text{ then } (()) \odot H, () \rangle \text{ else } ((()) \odot \top, ()) \rangle \text{ in } \text{publish} (\text{labelOf} (\text{fst} y) == H)
  \]

6 This Fixes the Two Problems from Step 2

6.1 The “Infectious pc” Problem Fixed

\[
\text{empty} \vdash H\langle \text{if } (\text{true} \odot H) \text{ then } (\text{true}, (\text{false}, ())) \text{ else } () \rangle, \bot \downarrow (\text{true} \odot \bot, (\text{false} \odot \bot, () \odot \bot) \odot H, \bot
\]

6.2 The “Poison Pill” Problem Fixed

• Labels are now public.

• Critical components can use label\_Of to protect themselves from “poison pills”.

• IFC violations no longer need to be fatal errors (still a lot of care needs to be used when adding exceptions, they don’t interact too well with brackets).