History based flow analysis in the lambda calculus

Tomasz Blanc
Jean-Jacques Lévy

INRIA Rocquencourt

IISc
February 8, 2006
(work in progress)
Plan

1. Dependency calculi
2. Stack inspection
3. History-based stack inspection
4. Confluency
5. Labeled lambda-calculus
6. Types
Many calculi exist since [76, Denning’s]:

- [97 Biswas], [97 Abadi, Lampson, J JL] dependency calculus for *makefiles*
- [99 Abadi, Banerjee, Heintze, Riecke] Dependency core calculus
- [00 Boudol, Castellani] Imperative programs
- …type checking + type inference

Non interference theorems.
Non interference

- $M$ public (low), $A$ is private (high)
- $M \rightarrow V$, $V$ value
- no leak of $A$ in $V$
- $M = C[A] \rightarrow V$ implies $C[B] \rightarrow V$
All (but first) are based on type theory and non-interference.

Is there an “untyped” theory?

Is non-interference wrt “security levels” the only property?
Stack inspection supports two sets of permissions:

- **dynamic permissions** \( D \)
- **static permissions** \( S \)

Reduction \( \rightarrow_S^D \) is parameterized by \( D \) and \( S \)
Stack inspection (2/5)

Language

\[ R, S, D ::= \]

\[ M, N ::= \]
\[ x | \lambda x. M | MN \]
\[ R[M] \]
\[ \text{grant } R \text{ in } M \]
\[ \text{test } R \text{ then } M \text{ else } N \]

\[ V ::= \lambda x. M \]

permissions set
expression
\( \lambda \)-expression
framed expression
permission grant
permission test
value

Reductions

- call-by-value

\[
\begin{align*}
M_1 & \rightarrow_S^{SD} M'_1 \\
M_1 M_2 & \rightarrow_S^{SD} M'_1 M_2 \\
M_2 & \rightarrow_S^{SD} M'_2 \\
V_1 M_2 & \rightarrow_S^{SD} V_1 M_2
\end{align*}
\]

\[ (\lambda x. M)V \rightarrow_S^{SD} M\{x := V\} \]
Stack inspection (3/5)

- permission rules
  
  **[CtxFrame]**
  \[
  M \xrightarrow{R \cap R} M' \\
  R[M] \xrightarrow{S \cap \cap} R[M']
  \]

  **[CtxGrant]**
  \[
  M \xrightarrow{S \cup (R \cap S)} M' \\
  \text{grant } R \text{ in } M \xrightarrow{S \cap \cap} \text{grant } R \text{ in } M'
  \]

  **[RedFrame]**
  \[
  R[V] \xrightarrow{S \cap \cap} V
  \]

  **[RedGrant]**
  \[
  \text{grant } R \text{ in } V \xrightarrow{S \cap \cap} V
  \]

  **[RedTest]**
  \[
  \text{test } R \text{ then } M_{\text{true}} \text{ else } N_{\text{false}} \xrightarrow{S \cap \cap} M_{R \subseteq D}
  \]

- \(\cup, \cap, \subseteq\) are operations on permissions
- values are transparent for permissions
- static permission does not propagate in framed expressions
- stack inspection is a simple “untyped” calculus
Example with Java-like programs

class Applet { // -----------untrusted
    public static void main (String[ ] args) {
        NaiveLibrary.cleanUp ("/etc/passwd");
    }
}

class NaiveLibrary { // -----trusted
    static void cleanUp (String s) {
        File.delete (s);
    }
}

class File { // ---------trusted
    static void delete (String s) {
        FileIOPermission p = new FileIOPermission(s);
        p.checkDelete();
        System.deleteFile(s);
    }
}

check fails with stack inspection since
Applet[main(Lib[cleanUp(Sys[test FileDelete in delete(s) else fail]])])]
Applet ∩ Sys = ∅
stack inspection provides a weak non-interference property
⇒ static analyzer for C# libraries
[04, Blanc, Fournet, Gordon]
with long proofs for soundness
[03, Abadi, Fournet] informal description of history-based stack inspection solving 2 examples:

**BadPlugin example ↔ untrusted values**

```java
class NaiveProgram {
   public static void main (String[] args) {
      String s = BadPlugin.tempFile ();
      NaiveLibrary.cleanUp (s);
   }
}

class NaiveLibrary {
   static void cleanUp (String s) {
      File.delete (s);
   }
}

class BadPlugin {
   static String tempFile () {
      return "/etc/passwd";
   }
}
```

does not fit in stack inspection since values are transparent for permissions
Chinese Wall: B should not access to private information of A and conversely

```java
public class Customer {
    int examine () {
        ...
        if (shouldConsiderA) {
            Contractor a = new companyA();
            return a.offer();
        }
    }

    static public void main (String[ ] args) {
        int offer = examine ();
        Contractor b = new companyB();
        // ---------raises exception if any B code has run
    }
    ...
}
```

does not fit in stack inspection since not in a chain of function calls
Non interference between sub-expressions

- $A$ and $B$ are two different parties
- $M \rightarrow V$, $V$ value
- no interaction between $A$ and $B$ is necessary to produce $V$
- $V$ may contain $A$ and $B$
- interference theorem much harder to state

What is interaction between $A$ and $B$?
confluence \equiv \text{independence} \text{ of evaluation strategy} \\
\Rightarrow \text{equational theory} \Rightarrow \text{simplicity} \\
confluence \Rightarrow \text{static analysis by abstract interpretation} \\
dynamic \text{ information is inherently non confluent} \\
as for the dynamically-scoped \(\lambda\)-calculus

\[
(\lambda x.\lambda y.(\lambda x.\lambda y.x)y x) ab \rightarrow \ldots \rightarrow (\lambda x.\lambda y.x) ba \rightarrow a \\
(\lambda x.\lambda y.(\lambda x.\lambda y.x)y x) ab \rightarrow \ldots \rightarrow (\lambda x.\lambda y.y) ab \rightarrow b
\]

stack inspection is not confluent

when \( \text{FileIO} \subseteq \text{Sys} \)
\[
\text{Sys}[(\lambda x. \text{Applet}[x] V) (\text{test FileIO in } (\lambda x.x)(\lambda x.a) \text{ else } \text{fail})] \\
\rightarrow \ldots \rightarrow a \quad \text{Call by Value} \\
\rightarrow \ldots \rightarrow \text{fail} \quad \text{Call by Name}
\]
The labeled $\lambda$-calculus (1/7)

Language

$$\alpha, \beta, \gamma ::= a \mid \lceil \alpha \rceil \mid \lfloor \alpha \rfloor \mid \alpha\beta$$

labels

atomic name

compound name

$\epsilon$ empty string

$M, N ::= x \mid (\lambda x.M) \mid (MN) \mid M^\alpha \lambda$-expression

labeled expression

Exponent Rules

$$(M^\alpha)^\beta = M^{\beta\alpha} \quad M^\epsilon = M \quad \lceil \epsilon \rceil = \lfloor \epsilon \rfloor = \epsilon$$

Reduction

$$(\lambda x.M)^\alpha N \longrightarrow (M\{x := N^{\lfloor \alpha \rfloor}\})^{\lceil \alpha \rceil}$$

$$x^\alpha\{x := P\} = P^\alpha$$

$$y^\alpha\{x := P\} = y^\alpha$$

$$(\lambda y.M)^\alpha\{x := P\} = (\lambda y.M\{x := P\})^\alpha$$

$$(MN)^\alpha\{x := P\} = (M\{x := P\}N\{x := P\})^\alpha$$
The labeled $\lambda$-calculus (2/7)

Graphically

- $M$ is sandwiched by $[\alpha]$ and $[\alpha]$
  $\Rightarrow$ theory of balanced paths [94, Asperti, Laneve, Guerrini, Mairson, Danos, Reigner, ...]

$\leftrightarrow$ Girard’s geometry of interaction
The labeled $\lambda$-calculus is confluent (thanks to exponent rules)

the labeled $\lambda$-calculus tracks history of redexes (redex families)

the labeled $\lambda$-calculus corresponds to the event structure of redexes

$\Rightarrow$ the labeled $\lambda$-calculus is a good candidate for a confluent equational theory of flow analysis (lattice of derivations, stability, . . . )
e.g. dependency calculus for makefiles uses a tiny subset
The labeled $\lambda$-calculus (4/7)

- If $M \rightarrow V$, there is a unique minimum $A$ of $M$ such that $A \rightarrow V$ [stability thm]

- If $C[M] \rightarrow V$, there is a unique minimum prefix $A$ of $M$ such that $C[A] \rightarrow V'$ [corollary of stability thm]

- [97, Abadi, Lampson, JJL] compute minimum prefix by:
  - Mark all subexpression with different atomic label;
  - perform $M \rightarrow V$
  - erase part of $M$ not in $V$.

- simple and good for incremental computations (Vista)

- also characterizes non interference when $M = C[A]$ [99, Conchon, Pottier]
The labeled $\lambda$-calculus (5/7)

- the labeled $\lambda$-calculus is good for tracing interactions.
- to build the Chinese Wall:
  Let $M = C[A; B] \rightarrow V$. Let mark subexpressions in $A$ with $a$, and in $B$ with $b$.
  There should not be any label $\gamma$ in $V$ such that $\gamma = \cdots [a \cdots b] \cdots$ or $\gamma = \cdots [a \cdots b] \cdots$.

- sets as labels

$$
\begin{align*}
\llbracket a \rrbracket_i &= \{a\} \\
\llbracket \alpha \beta \rrbracket_i &= \llbracket \alpha \rrbracket_i \cup \llbracket \beta \rrbracket_i \\
\llbracket \llbracket \alpha \rrbracket \rrbracket_1 &= \llbracket \llbracket \alpha \rrbracket \rrbracket_1 = \llbracket \llbracket \alpha \rrbracket_0 \\
\llbracket \llbracket \alpha \rrbracket \rrbracket_0 &= \llbracket \llbracket \alpha \rrbracket \rrbracket_0 = \llbracket \alpha \rrbracket_0
\end{align*}
$$

where $i = 0, 1$ and $\{\emptyset\} = \emptyset$

- $P(\alpha) = \neg \exists a \exists b. \ a, b \in X \in \llbracket \alpha \rrbracket_1$
the labeled \( \lambda \)-calculus restricted by a predicate \( \mathcal{P} \)

Reduction \( (\lambda x. M)^\alpha N \rightarrow (M \{ x := N[^{\alpha}] \})[^{\alpha}] \) when \( \models \mathcal{P}(\alpha) \)

the labeled \( \lambda \)-calculus restricted by \( \mathcal{P} \) is still confluent for any \( \mathcal{P} \).
Let $\alpha < \beta$ be the causality relation:

$\alpha < \lceil \alpha \rceil \quad \alpha < \lfloor \alpha \rfloor$

$\alpha < \beta \implies \alpha < \gamma \beta \delta$

Chinese Wall for independent spinoffs of $A$

$P(\alpha) = \neg (\exists \beta \exists \gamma \, \beta \not< \gamma \land \gamma \not< \beta \land A < \beta < \alpha \land A < \gamma < \alpha)$

$\beta \not< \gamma$ is not so easy to test

equality between subtrees of the $\alpha$ tree

simpler versions ? [Tomasz Blanc]

from labeled $\lambda$-calculus towards DCC (Dependency Core Calculus) or other flow calculi with types ???

deontic logic ?
Type systems and labels

[Sub]
\[ \Gamma \vdash M : t \quad t \leq t' \]
\[ \Gamma \vdash M : t' \]

[Var]
\[ x \in \text{domain} (\Gamma) \]
\[ \Gamma \vdash x : \Gamma (x) \]

[Lambda]
\[ \Gamma, x : t \vdash M : t' \]
\[ \Gamma \vdash \lambda x.M : t \rightarrow t' \]

[App]
\[ \Gamma \vdash M : t \quad \alpha \rightarrow t' \quad \Gamma \vdash N : [\alpha] \circ t \]
\[ \Gamma \vdash MN : [\alpha] \circ t' \]

[Exponent]
\[ \Gamma \vdash M : t \]
\[ \Gamma \vdash M^\alpha : \alpha \circ t \]

- pushing labels on types (with \( \leq \))
- Infers [02, Pottier, Simonet]
Conclusion

- Stack inspection is **not** static analysis
- Dynamic checks support **finer tests** for security
- Attempts for mixing history and stack inspection
- Confluency is a **hint** for “good” calculi
  - Stack inspection is **not a good calculus**
  - Finer **flow analysis**
- Statically scoped information (**static permissions** of stack inspection) should be carried by the **labeled \( \lambda \)-calculus**. (e.g. Chinese Wall)
- Abstract interpretation of labeled lambda calculus?