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

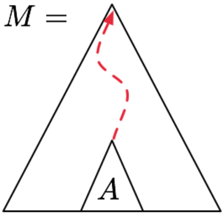
Dependency calculi (1/2)

Many calculi exist since [76, Denning's]:

- [97 Biswas], [97 Abadi, Lampson, JJL]
  dependency calculus for *makefiles*
- [98-00 Pottier, Simonet, Heintze, Riecke]
  type theory with security information à la
  [97 Volpano, Smith]
  for ML-like programs.
- [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)
- $M \twoheadrightarrow V$, V value
- no leak of A in V
- $M = C[A] \twoheadrightarrow V$ implies $C[B] \twoheadrightarrow V$

Dependency calculi (2/2)

- 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 (1/5)

[Fournet, Gordon, POPL'02]

- flow analysis based on procedure calls
- JVM $+$ CLR security manager $\Rightarrow$ stack inspection

Stack inspection supports two sets of permissions:

- dynamic permissions D
- static permissions S
- reduction $\longrightarrow_D^S$ is parameterized by D and S

Stack inspection (2/5)

Language

R, S, D	$::=$		permissions set
M, N	$::=$		expression
	$x \mid \lambda x.M \mid MN$		λ-expression
	$R[M]$		framed expression
	$\texttt{grant}\, R\, \texttt{in}\, M$		permission grant
	$\texttt{test}\, R\, \texttt{then}\, M\, \texttt{else}\, N$		permission test
V	$::=$	$\lambda x.M$	value

Reductions

- call-by-value

$$\frac{M_1 \longrightarrow_D^S M_1'}{M_1 M_2 \longrightarrow_D^S M_1' M_2} \qquad \frac{M_2 \longrightarrow_D^S M_2'}{V_1 M_2 \longrightarrow_D^S V_1 M_2}$$

$$(\lambda x.M)V \longrightarrow_D^S M\{x := V\}$$

Stack inspection (3/5)

- permission rules

[*CtxFrame*]

$$\frac{M \longrightarrow^{R}_{D\cap R} M'}{R[M] \longrightarrow^{S}_{D} R[M']}$$

[*CtxGrant*]

$$\frac{M \longrightarrow^{S}_{D\cup(R\cap S)} M'}{\texttt{grant}\, R\ \texttt{in}\ M \longrightarrow^{S}_{D} \texttt{grant}\, R\ \texttt{in}\ M'}$$

[*RedFrame*]

$$R[V] \longrightarrow^{S}_{D} V$$

[*RedGrant*]

$$\texttt{grant}\, R\ \texttt{in}\ V \longrightarrow^{S}_{D} V$$

[*RedTest*]

$$\texttt{test}\, R\ \texttt{then}\ M_{true}\ \texttt{else}\ N_{false} \longrightarrow^{S}_{D} M_{R\subseteq D}$$

- $\cup$, $\cap$, $\subseteq$ are operations on permissions
- values are transparent for permissions
- static permission does not propagate in framed expressions
- stack inspection is a simple “untyped” calculus

Stack inspection (4/5)

- Example with Java-like programs

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

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

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

- check fails with stack inspection since
  $Applet[main(Lib[cleanUp(Sys[\texttt{test}\ FileDelete\ \texttt{in}\ delete(s)\ \texttt{else}\ fail])])]$
  $Applet \cap Sys = \emptyset$

Stack inspection (5/5)

- stack inspection provides a weak non-interference property
- $\Rightarrow$ static analyzer for C# libraries
  `[04, Blanc, Fournet, Gordon]`
- with long proofs for soundness

History-based stack inspection (1/2)

- [03, Abadi, Fournet] informal description of history-based stack inspection solving 2 examples:
- BadPlugin example ↔ untrusted values

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

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

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

- does not fit in stack inspection
  since values are transparent for permissions

History-based stack inspection (2/2)

- Chinese Wall: B should not access to private information of A and conversely

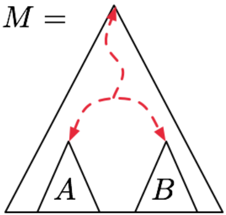
```
public class Customer {
  int examine () {
    ...
    if (shouldConsiderA) {
      Contractor a = new companyA();
      return a.offer();
  } }

  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
- $M \twoheadrightarrow V$, 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?

Dependency calculi and Confluency

- confluency $\equiv$ independence of evaluation strategy
  $\Rightarrow$ equational theory $\Rightarrow$ simplicity
- confluency $\Rightarrow$ static analysis by abstract interpretation
- dynamic information is inherently non confluent
  as for the dynamicaly-scoped λ-calculus

$$(\lambda x.\lambda y.(\lambda x.\lambda y.x)yx)ab \longrightarrow \ldots \longrightarrow (\lambda x.\lambda y.x)ba \longrightarrow a$$
$$(\lambda x.\lambda y.(\lambda x.\lambda y.x)yx)ab \longrightarrow \ldots \longrightarrow (\lambda x.\lambda y.y)ab \longrightarrow b$$

- stack inspection is not confluent

when $FileIO \subseteq Sys$
$Sys[(\lambda x.\,Applet[x]V)(\texttt{test}\ FileIO\ \texttt{in}\ (\lambda x.x)(\lambda x.a)\ \texttt{else}\ fail)]$
$\longrightarrow \ldots \longrightarrow a$ Call by Value
$\longrightarrow \ldots \longrightarrow fail$ Call by Name

The labeled λ-calculus (1/7)

Language

$\alpha, \beta, \gamma ::=$		labels
	$a \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid$	atomic name
	$\alpha\beta$	compound name
ϵ	empty string	
M, N	$::=$	labeled expression
	$x \mid (\lambda x.M) \mid (MN) \mid M^\alpha$	λ-expression

Exponent Rules

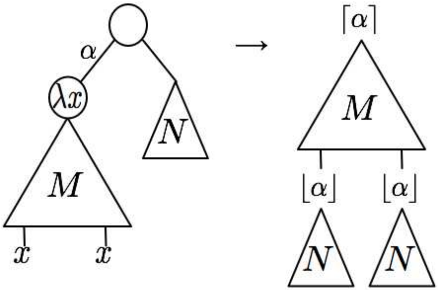
$$(M^\alpha)^\beta = M^{\beta\alpha} \qquad M^\epsilon = M \qquad \lceil\epsilon\rceil = \lfloor\epsilon\rfloor = \epsilon$$

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

$$\begin{aligned}
&x^\alpha\{x := P\} = P^\alpha \\
&y^\alpha\{x := P\} = y^\alpha \\
&(\lambda y.M)^\alpha\{x := P\} = (\lambda y.M\{x := P\})^\alpha \\
&(MN)^\alpha\{x := P\} = (M\{x := P\}N\{x := P\})^\alpha
\end{aligned}$$

The labeled λ-calculus (2/7)

Graphically



- *M* is sandwiched by $\lceil \alpha \rceil$ and $\lfloor \alpha \rfloor$
  $\Rightarrow$ theory of balanced paths [94, Asperti, Laneve, Guerrini, Mairson, Danos, Reigner, ...]
  $\leftrightarrow$ Girard's geometry of interaction

The labeled λ-calculus (3/7)

- the labeled λ-calculus is confluent
  (thanks to exponent rules)
- the labeled λ-calculus tracks history of redexes
  (redex families)
- the labeled λ-calculus corresponds to the event structure of redexes
- $\Rightarrow$ the labeled λ-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 λ-calculus (4/7)

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

- [97, Abadi, Lampson, JJL] compute minimum prefix by:
  - Mark all subexpression with different atomic label;
  - perform $M \twoheadrightarrow V$
  - 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 λ-calculus (5/7)

- the labeled λ-calculus is good for tracing interactions.
- to build the Chinese Wall:
  Let $M = C[A; B] \twoheadrightarrow V$. Let mark subexpressions in A with a, and in B with b.
  There should not be any label γ in V such that $\gamma = \cdots \lfloor a \cdots b \rfloor \cdots$ or $\gamma = \cdots \lceil a \cdots b \rceil \cdots$.
- sets as labels

$$
\begin{aligned}
[\![a]\!]_i &= \{a\} \\
[\![\alpha\beta]\!]_i &= [\![\alpha]\!]_i \cup [\![\beta]\!]_i \\
[\![\lceil\alpha\rceil]\!]_1 &= [\![\lfloor\alpha\rfloor]\!]_1 = \{[\![\alpha]\!]_0\} \\
[\![\lceil\alpha\rceil]\!]_0 &= [\![\lfloor\alpha\rfloor]\!]_0 = [\![\alpha]\!]_0
\end{aligned}
$$

  where $i = 0, 1$ and $\{\emptyset\} = \emptyset$
- $\mathcal{P}(\alpha) = \neg\, \exists a \exists b.\ a, b \in X \in [\![\alpha]\!]_1$

The labeled λ-calculus (6/7)

- the labeled λ-calculus restricted by a predicate $\mathcal{P}$
  Reduction $(\lambda x.M)^{\alpha} N \longrightarrow (M\{x := N^{\lfloor \alpha \rfloor}\})^{\lceil \alpha \rceil}$ when $\models \mathcal{P}(\alpha)$

- the labeled λ-calculus restricted by $\mathcal{P}$ is still confluent for any $\mathcal{P}$.

The labeled λ-calculus (7/7)

- Let $\alpha < \beta$ be the causality relation:
  $\alpha < \lceil \alpha \rceil \qquad \alpha < \lfloor \alpha \rfloor$
  $\alpha < \beta \;\Rightarrow\; \alpha < \gamma\beta\delta$
- Chinese Wall for independent spinoffs of A
  $\mathcal{P}(\alpha) = \neg(\exists \beta \, \exists \gamma \; \beta \not\leq \gamma \wedge \gamma \not\leq \beta \wedge A < \beta < \alpha \wedge A < \gamma < \alpha)$
- $\beta \not\leq \gamma$ is not so easy to test
  equality between subtrees of the α tree
- simpler versions ? [Tomasz Blanc]
- from labeled λ-calculus towards DCC (Dependency Core Calculus) or other flow calculi with types ???
- deontic logic ?

Type systems and labels

[*Sub*]
$$\frac{\Gamma \vdash M : t \quad t \leq t'}{\Gamma \vdash M : t'}$$

[*Var*]
$$\frac{x \in domain(\Gamma)}{\Gamma \vdash x : \Gamma(x)}$$

[*Lambda*]
$$\frac{\Gamma, x : t \vdash M : t'}{\Gamma \vdash \lambda x.M : t \longrightarrow t'}$$

[*App*]
$$\frac{\Gamma \vdash M : t \xrightarrow{\alpha} t' \quad \Gamma \vdash N : \lfloor \alpha \rfloor \circ t}{\Gamma \vdash MN : \lceil \alpha \rceil \circ t'}$$

[*Exponent*]
$$\frac{\Gamma \vdash M : t}{\Gamma \vdash M^{\alpha} : \alpha \circ t}$$

- 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 λ-calculus. (e.g. Chinese Wall)
- abstract interpretation of labeled lambda calculus?