A Monadic Analysis of Information Flow Security with Mutable State

Karl Crary, Aleksey Kliger, Frank Pfenning

Carnegie Mellon University

FCS Workshop
Turku, Finland, July 2004
The ConCert Project

- Certified distributed computation
- Technical basis
  - Typed assembly language (TAL, TALT)
  - Certifying compilation (TILT, PCC)
- Some technical challenges
  - Types for distributed computation
  - Resource bound certification
  - Architecture verification
  - Information flow
Information Flow in TAL

- Typed assembly language
  - Imperative
  - Functional
  - Sequentialized

- Abstract to high-level functional language
  - Capture analogous features
  - Easier to design, prove correct, understand
  - Future work: transfer to TAL
Language Overview

• Information flow only through store
• Effects encapsulated in monad
• Other computations and values remain pure
• Monad and locations indexed by security levels
• Subtyping to avoid security level coercions
• Allow upcalls via informativeness judgment
Outline

- Monadic encapsulation of effects
- Information flow and store
- Upcalls and informativeness
- Proof of non-interference
- Embedding value-oriented languages
Pure Functional Core

- Standard constructs

Types \( A ::= \) bool | 1 | \( A \rightarrow B \) | …

- Standard judgments
  - Typing \( \Gamma \vdash M : A \)
  - Value \( M \ val \) (write \( V \) for values)
  - Reduction \( M \rightarrow M' \)

- Call-by-value (could be by name or by need)
- Curry-Howard isomorphism (omit recursion)
Sample Rules: Functions

- **Typing**

\[
\frac{\Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x:A.M : A \to B} \to I
\]

- **Evaluation**

\[
\lambda x:A.M \text{ val} \quad \frac{M \to M'}{M \ N \to M' \ N} \quad (\lambda x:A.M) \ V \to M[V/x]
\]
Monadic Encapsulation

- New type \( \bigcirc A \) for effectful computations
- New syntactic category: expressions

Terms

\[ M ::= \ldots \mid \text{val } E \]

Expressions

\[ E ::= \text{let val } x = M \text{ in } E \mid M \]

- Expressions include terms
- Sequencing of effects via \text{let val}
- Further expressions for specific monads
Lax Typing

- Lax typing $\Gamma \vdash E \div A$

\[
\begin{align*}
\Gamma \vdash M : A \\
\Gamma \vdash M \div A
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash E \div A \\
\Gamma \vdash \text{val } E : \circ A
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash M : \circ A \\
\Gamma, x : A \vdash E \div C
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash \text{let val } x = M \text{ in } E \div C
\end{align*}
\]

- Restriction on elimination enforces sequencing
- Related to \textit{lax logic} by Curry-Howard isomorphism
Operational Semantics

- Computation steps \((H, E) \rightarrow (H', E')\) for store \(H\)

\[
\begin{align*}
\text{val } E \text{ val} & \quad M \rightarrow M' \\
(H, M) \rightarrow (H, M') \\
M \rightarrow M' & \\
(H, \text{let val } x = M \text{ in } F) \rightarrow (H, \text{let val } x = M' \text{ in } F) \\
(H, E) \rightarrow (H', E') & \\
(H, \text{let val } x = \text{val } E \text{ in } F) \rightarrow (H', \text{let val } x = \text{val } E' \text{ in } F) \\
V \text{ val} & \\
(H, \text{let val } x = \text{val } V \text{ in } F) \rightarrow (H, F[V/x])
\end{align*}
\]
Security Levels

- Fixed lattice \( a \sqsubseteq b \)
- Operations \( \bot, \top, \sqcap, \sqcup \)
- Store locations \( l \) have security level \( a \), type \( A \) (write: \( l_a^A \), omit when clear)
- Computation \( E \div (r, w) A \) has security levels
  - \( r \): can read only at \( r \) or below
  - \( w \): can write only at \( w \) or above
  - operation level \( o = (r, w) \) for \( r \sqsubseteq w \)
- Terms \( M : A \) have no effect, no security level
Store locations $l^A_a$ with intrinsic security level $a$

- Store locations are terms (no effect)
- Store locations are values

\[ l^A_a \rightarrow \text{val} \]

- Stores uniquely bind locations to values

\[
\text{Store } H \ ::= \cdot \mid H, l^A_a \rightarrow V
\]
Allocation, Reading, Writing

- Assign most precise type; others by subtyping
- Write $E \div (r, w) A$ for readability
- Allocation neither reads nor writes

\[
\Gamma \vdash l^A_a : \text{ref}_a A \\
\Gamma \vdash \text{ref}_a M \div (\bot, \top) \text{ref}_a A
\]

- Reading and writing are effects

\[
\Gamma \vdash M : \text{ref}_a A \\
\Gamma \vdash !M \div (a, \top) A \\
\Gamma \vdash M := N \div (\bot, a) 1
\]
Subtyping

- \( A \leq B \) : \( A \) is subtype of \( B \)
- \( o \preceq p \) : \( o \) is less strict than \( p \)
- Subsumption rules

\[
\frac{\Gamma \vdash M : A \quad A \leq B}{\Gamma \vdash M : B}
\]

\[
\frac{\Gamma \vdash E \div_o A \quad o \preceq p}{\Gamma \vdash E \div_p A}
\]

\[
\frac{\Gamma \vdash E \div_o A \quad A \leq B}{\Gamma \vdash E \div_o B}
\]
Variance

- Recall $E \div (r, w) A$
  - reads only below $r$
  - writes only above $w$

- Co-variant in read, contra-variant in write

\[
\frac{r \subseteq r'}{r, w) \leq (r', w')}
\]

\[
\frac{w' \subseteq w}{A \leq B}
\]

\[
\frac{o \leq p}{\bigcirc_o A \leq \bigcirc_p B}
\]

- $\text{ref}_a A$ is non-variant (paper: $\text{ref}_{r, A}$ and $\text{ref}_{w, A}$)
- Other subtyping standard
Operational Semantics Revisited

- Standard rules for reduction with store
- Example: allocation

\[
\begin{align*}
M & \rightarrow M' \\
(H, \text{ref}_a M) & \rightarrow (H, \text{ref}_a M') \\
V \text{ val} & \quad l_a \not\in \text{dom}(H) \\
(H, \text{ref}_a V) & \rightarrow ((H, l \mapsto V), l)
\end{align*}
\]
Lax Typing Revisited

- Lax security typing $\Gamma \vdash E \div_o A$

$\frac{\Gamma \vdash M : A}{\Gamma \vdash M \div (\bot, \top) A}$

$\frac{\Gamma \vdash E \div_o A}{\Gamma \vdash \text{val } E : \bigcirc_o A}$

$\bigcirc I$

$\frac{\Gamma \vdash M : \bigcirc_o A}{\Gamma \vdash \text{let val } x = M \text{ in } E \div_o C}$

$\bigcirc E$

- $(\bot, \top)$ is minimal for $\preceq$
Upcalls

- Consider a call of $E$ at high security from within $F$ at low security

\[
E \div (\top, \top) \ 1 \\
\vdash z : 1 \ \Rightarrow F \div (\bot, \bot) \ 1 \\
\text{let val } z = \text{val } E \text{ in } F \div (?, \bot) \ 1
\]

- Current rules force $? = \top$
- Does $E$ leak information?
- Depends of type of returned value (here, 1)
Informativeness

- $A \uparrow r$  
  A is informative only at $r$ and above
- Use to demote reading level of expressions

$$
\Gamma \vdash E \div (r, w) A \quad A \uparrow r
\quad
\Gamma \vdash E \div (\bot, w) A
$$

- Some rules

$$
1 \uparrow r\quad B \uparrow b
\quad
A \rightarrow B \uparrow b
$$
Informativeness of Computations

• Storage locations

\[
\begin{align*}
\text{ref}_b A & \rightarrow b \\
\text{ref}_b A & \rightarrow a
\end{align*}
\]

• Computations

\[
\begin{align*}
A & \rightarrow a \\
\bigcirc_{(r,w)} A & \rightarrow w \sqcap a
\end{align*}
\]
General Information Laws

• Contra-variant in security level

\[
\begin{align*}
A \not\rightarrow \bot & \quad A \not\rightarrow a \quad b \sqsubseteq a \\
A \not\rightarrow b & \quad A \not\rightarrow c \\
A \not\rightarrow b \sqcup c
\end{align*}
\]

• Now can type \textit{untilFalse} : \(\bigcirc_{(T,T)} \text{Bool} \rightarrow \bigcirc_{(\bot,T)} 1\) [see paper]

• Do not consider termination channel
Theorems

• Write $\vdash H$ if store is well-typed
• Write $\vdash (H, E) \div_o A$ if $\vdash H$ and $\vdash E \div_o A$
• Language so far satisfies
  • *Preservation*: If $\vdash (H, E) \div_o A$, and $(H, E) \rightarrow (H', E')$, then $\vdash (H', E') \div_o A$.
  • *Progress*: If $\vdash (H, E) \div_o A$ then either $E = V$ for $V$ val or $(H, E) \rightarrow (H', E')$ for some $(H', E')$
  • *Non-interference*: “Computations at low security cannot observe high-security values”
Sketch of Non-Interference

- Define *in-view locations* for level \( \zeta \):

  \[
  \downarrow (\zeta) = \{ l_a \mid a \sqsubseteq \zeta \}
  \]

- Define equivalence on in-view locations:

  \[
  H_1 \approx_{\zeta} H_2 \quad \text{and} \quad (H_1, E_1) \approx_{\zeta} (H_2, E_2) \div_o A
  \]

- **Theorem:** If \( \vdash H \) and \( x:A \vdash E \div_{(r,w)} B \) and \( V_1 \approx_r V_2 : A \) then if \( (H, E[V_1/x]) \rightarrow^* S_1 \) and \( (H, E[V_2/x]) \rightarrow S_2 \) then \( S_1 \approx_r S_2 \div_{(r,w)} B \).

- **Proof:** Syntactic, using Church-Rosser modulo in-view equivalence with respect to \( r \).
Related Work

- Information flow inference for ML [Pottier & Simonet’03]
  - Any term may have an effect
  - Emphasis on inference
  - Here: monadic encapsulation, checking

- Dependency Core Calculus (DCC) [Abadi, Banerjee, Heintze, Riecke’99]
  - Monads for sealing values, not state
  - Protectedness $\sim$ informativeness
Related Work

- $\lambda_{SEC}^{REF}$ [Zdancewic’02]
  - Security levels for values, not locations
  - Can be mapped to our language [see paper]

- Information flow for $\pi$-calculus [Honda&Yoshida’02]
  - Different computational setting
  - Tampering levels $\sim$ informativeness

- Domain separation [Harrison,Tullsen,Hook’03]
  - State insulation via monads
  - No interaction between monads
Future Work

- Additional effects (I/O, control effects)
- Information flow in TAL (register re-use)
- Decomposing the monad into □, ♦
  [Pf.&Davies’01]
- Dependent type theory with information flow
Summary

- Type system for information flow
  - Higher-order functional language
  - Store monad, indexed by operation levels
  - Security levels for locations, not values
- Conservative over base language
- Upcalls permitted via informativeness
- Preservation, progress, non-interference