Towards a Realistic Formal Model for Distributed Systems

- Calculus [Milner, Parrow, Walker 89]
  Channels/Names

\[
a(x) . P \mid \overline{a}(b) \rightarrow P[b/x]
\]

Input Output

- Real World (cf. Applet Passing in Java)

Diagram:
- Server
- Client:
  - Download
  - Software
- Client:
  - Software
Higher-Order \( \pi \)-Calculus

[Sangiorgi 93]
CML, Facile, LLinda, ...

\[ \forall v (\lambda x. Q) v \rightarrow Q[\nu/x] \]

\[ \forall v \bar{a}[P] \mid a(x). run \ x \rightarrow run \bar{P} \rightarrow P \]

where
\[ \text{thunk} \quad \bar{P} = \lambda(). P \]
\[ \text{run} \quad = \lambda x. x() \]

---

Server

Client

Client†
Aims of Types/Typechecking

- Using Types to control the effects of Mobile Code / Processes

- Host refuses to execute incoming code unless it conforms to predetermined access policy

\[ a(x: T). P \]

Q \rightarrow Type

STOP
Existing \( \lambda / \pi \) Typing System

\[ 1993 - 2000 \]

Before [YH00]

\[ \lambda \rightarrow + \Pi.I0 \]

\[ \tau ::= \text{unit} \mid \text{nat} \mid \tau \rightarrow \rho \mid 6 \mid \text{proc} \]

Value

Term

\[ 6 ::= (\tau)^I \mid (\tau)^O \mid (\tau)^{I_0} \]

Channel

Input

Output

Input-Output

Typing Processes .... Too Simple

\[ \Gamma \vdash P : \text{proc} \]

e.g.

\[ \Gamma \vdash P : \text{proc} \quad \Gamma \vdash Q : \text{proc} \]

\[ \Gamma \vdash P \| Q : \text{proc} \]

cf.

\[ \Gamma \vdash M : \tau \rightarrow \rho \]
Problem

\[ c(\langle X : \text{proc} \rangle). \text{run } X \]

where \( \langle \text{proc} \rangle = \text{unit} \rightarrow \text{proc} \)

Any Process is well-come

\[ P \quad a(X). \overline{b} \langle X \rangle \]

\[ Q \quad b(X). \overline{a} \langle X \rangle \]

\[ R \quad d \langle \text{string} \rangle \]

We are very different

I will give some technicality
Assigning Types to Processes

\[ \Gamma \vdash P : [\Delta] \]

Example

\[ P \vdash a(x) \cdot \overline{b}\langle x\rangle : [a:(\tau)^1, b:(\tau)^0] \]

\[ Q \vdash b(x) \cdot \overline{a}\langle x\rangle : [a:(\tau)^0, b:(\tau)^1] \]

\[ R \vdash d\langle \text{string}\rangle : [\ d:(\text{string})^0] \]

\[ c(X: \langle a:(\tau)^1, b:(\tau)^0 \rangle). \text{run} \ X \]
History of Types in HO\(\pi\)/\(\pi\)

**HO\(\pi\)**

- 93 HO\(\pi\) [Sangiorgi PhD]

**\(\pi\)**

- 89 \(\pi\)-Calculus
- 92 Polyadic \(\pi\)
- 93 IO-Subtyping
- Linear Typings
- Polymorphism
- Causality-based Typings

**Fine-Grained Types**

- 98 Fine-Grained Types [Yoshida and Hennessy]

**Other Mobility Types**

- 00 Seal Calculus
- \(\lambda\rightarrow+x\) Calculus
- Safe-Boxed Ambient

**Fully Abstraction**

- 01 PCF [TCAL 01]
- System F [FoSSacs 01]
- Control [CW 04]

**Secure Info Flow**

- 02 [ESOP 01, FoSSacs 02, POPL 02]

**Channel Dependency**

- 03 Existential Types

\(\Rightarrow\) Integration with Linearity

\(\Rightarrow\) Secure Info Role-Based Access Control
the Idea is simple but... quite a bit of work

[1998 - 2000 - 2003]

Channels appear both in Types and Processes

\[ F = \lambda x. \lambda (X : \langle x : (2) \circ, b : (2)^{\circ} \rangle). (\text{run } X \mid \overline{x} < 1 > \mid \overline{b} < 2 >) \]

\[ F a \rightarrow \lambda (X : \langle a : (2)^{\circ}, b : (2)^{\circ} \rangle). (\text{run } X \mid \overline{a} < 1 > \mid \overline{b} < 2 >) \]

\[ F b \rightarrow \lambda (X : \langle b : (2)^{\circ} \rangle). (\text{run } X \mid \overline{b} < 1 > \mid \overline{b} < 2 >) \]

⇒ Kinding / Dependency Types

[Yoshida · Hennessy 2000] Functional Channel Dependency

POPL 04 NEW Channel Dependency (Non-determinism)

POPL 04 NEW Existential Types (Scope-Opening)
Higher Order π-Calculus

Syntax

\[ P, Q ::= V \quad \text{value} \]
\[ \null \]
\[ 0 \quad \text{nil} \]
\[ \null \]
\[ \bar{u} \langle V_1, \ldots, V_m \rangle \quad \text{OUTPUT} \]
\[ \null \]
\[ u(x:Z_1, \ldots, x:Z_m)P \quad \text{INPUT} \]
\[ \null \]
\[ !P \quad \text{Replication} \]
\[ \null \]
\[ (va:e)P \quad \text{Restriction} \]
\[ \null \]
\[ PQ \quad \text{Application} \]
\[ \null \]
\[ V, W ::= \lambda (x:Z)P \quad \text{λ-abst} \]
\[ \null \]
\[ 1, 2, \ldots, (), \ldots \quad \text{constant} \]
\[ \null \]
\[ x, y, z, \ldots \quad \text{variables} \]
\[ \null \]
\[ a, b, c, \ldots \quad \text{channels/names} \]
Types

Term \( \mathcal{I} ::= \text{unit}, \text{nat} \)

\[ \mathcal{I} \rightarrow \mathcal{I} \]

\[ \Pi(x : \mathcal{I}) \mathcal{I} \]

\[ \left[ \Delta \right] \]

\[ 6 \]

functional dependency

process types

Channel \( \mathcal{6} ::= (\Pi (x : \mathcal{I}) \mathcal{I})^p \) dependency

\[ (\forall x : \mathcal{I} \mathcal{I})^p \]

existential

\[ \langle 6_1, 6_0 \rangle \]
Typing System \( \Gamma \vdash P : \Delta \)

**Zero**

\[
\frac{\Gamma \vdash \text{Env}}{
\Gamma \vdash 0 : [\_]
}\]

**Par**

\[
\frac{
\Gamma \vdash P : [\Delta] \quad \Gamma \vdash Q : [\Delta']
}{
\Gamma \vdash P \mathbin{\|} Q : [\Delta \cup \Delta']
}\]

**Res**

\[
\frac{
\Gamma, a : \! a \vdash P : [\Delta, a : \! a]
}{
\Gamma \vdash (\mathsf{va} : \! a) P : [\Delta]
}\]

**Rep**

\[
\frac{
\Gamma \vdash P : [\Delta]
}{
\Gamma \vdash ! P : [\Delta]
}\]
Existential Types for Scope Opening

Client (A) wishes to execute code P and to get ack \( \bar{c(v)} \) at the time P is executed at the remote location (B).

\[
\begin{align*}
&\forall c \quad b\langle P | \bar{c(v)} \rangle | c(x).R \\
&\text{code} \quad \text{ack} \quad \text{wait at } c
\end{align*}
\]

(1) \( c \) is private

(2) \( v \) must not be touched (i.e. compromised)
Existential Types for Scope Opening

(B)

\[ b(x).\text{run}\ x \]

(vc)

\[ \bar{b} \langle p \mid \bar{c} (\nu) \rangle \]

\[ c(y).R \]

\[ \text{private name} \]

(vc)

\[ \langle p \mid \bar{c} (\nu) \rangle \]

\[ c(y).R \]

(vc)

\[ \bar{c} (\nu) \]

\[ l \]

\[ c(y).R \]

previous system

[POPL 04]

channel existential types

\[ (\exists [x:6] \Gamma \Delta, x:6^-) ^I \]

\[ \text{anonymous channel of type 6} \]
Typing System for $\exists$

\[(\text{In}^3)\] 
\[\Gamma \vdash a : (\exists[x:6]z)^1\] 
\[\Gamma, \{x:6, x:z\} \vdash P \triangleright [\Delta, x:6]\] 
\[\Gamma \vdash a(x : \exists[x:6]z). P \triangleright [\Delta, a : (\exists[x:6]z)^1]\]

\[(\text{Out}^3)\] 
\[\Gamma \vdash a : (\exists[x:\overline{6}]z)^0\] 
\[\Gamma \vdash \{c, V\} : \exists[x:6]z\] 
\[\Gamma \vdash \overline{a}(V) \triangleright [a : (\exists[x:\overline{6}]z)^0, c:6]\] 
\[\Gamma \vdash \overline{a}(V) \triangleright [\overline{a} : (\exists[x:\overline{6}]z)^0, c:6]\] 
record a name
to be restricted

Proposition (Minimality)

\[\Gamma \vdash P \triangleright [\Delta] \Rightarrow \exists! \Delta' \subseteq \Delta\] 
s.t. \[\Gamma \vdash P \triangleright [\Delta']\]
Main Theorems

Subject Reduction
\[ \Gamma \vdash P : \Sigma, \ P \rightarrow P' \Rightarrow \Gamma \vdash P' : \Sigma \]

Type Safety
\[ \Gamma \vdash P : [\Delta] \Rightarrow P \xrightarrow{\Gamma, [\Delta]}_{\text{err}} \]
where \( P \xrightarrow{\Gamma, [\Delta]}_{\text{err}} \) means

P can use at most resources in \( \Delta \)

Consequence:
\[ \alpha (X : [\Delta^']). P \mid \overline{a} \langle R \rangle \xrightarrow{\Gamma, \pi}_{\text{err}} \]
if \( \Gamma \not\vdash R : [\Delta] \)
Encapsulation of Higher-Order Code by Hidden Name

Theorem
\[ \Gamma \vdash P : \Delta \] and \( \text{fv}(P) = \emptyset \)
and \[ \Gamma \vdash a : (\exists x : 6 \to x : 6) \]

Then \[ P \xrightarrow{a \langle \exists \langle v \rangle \rangle} P' \Rightarrow P' \Rightarrow \exists \langle v \rangle \]

Mobile Code bound by \( \exists \)-name is eventually returned to the sender without being touched by the receiver.
Secure Information Flow in HO\(\pi\)

why Fine Grained Process Types?

\[ a \cdot b \xrightarrow{\text{H}} 0 \]

\[ b \cdot a \xrightarrow{\text{L}} X \]

\[ b (\overline{a}) \xrightarrow{?\ ?} \text{a thunk of a L level program is transferred via H level channel} \]

\[ b (x) \cdot \text{run} X \mid a \cdot c \xrightarrow{?\ ?} X \]

\[ b (x) \cdot ( Ay. 0 ) X \mid a \cdot c \xrightarrow{?\ ?} O \]

\[ [\ ] \]
Theorem Non-Interference

\[ \Gamma \vdash P_{1,2} \triangleright [\Delta] \quad \text{s.t.} \quad \text{temp}(\Delta) \not\in S \]

Then \( P_1 \approx_s P_2 \)

Two processes with a secrecy level incompatible with \( S \) can be equated by \( \approx_s \)

Proof: By Type Preserving Translations into \( \Pi \text{sec} \) [Honda and Yoshida 2002]
Role-Based Access Control

Why Fine-Grained Process Types?

Extensible Architecture  SPIN/JVM/CLR

[1] Safe Access to Resources by Name-Space Control ⇒ Process Types


[3] Primitives to Alter one's own Access Right

\[ P ::= \ldots \mid \text{set(principal)} \{ P \} \mid !a^{\beta}(x:\tilde{x}), P \mid \{ P \}_{\text{principal}} \]

\{ \}

\text{thread running as principal}
Role-Based Access Control

Reduction
\[ \{\text{Set}(A) \{ P \} \}_B \rightarrow \{ P \}_A \]

\[ \{ a \langle V \rangle \}_A \mid \{ !a(x).R \}_B \]
\[ \rightarrow \{ R[\forall x] \}_A \cap _B \mid \{ !a(x).R \}_B \]

cf. Blacketing Condition via Linear/Affine
Theorem \ \text{Role-Based Resource Error Free}

\[ \Gamma \vdash P \supset [\Delta] \text{ then } P \xrightarrow{\text{Aerr}} \]

where

\[ \{ !a^c(x). R \}_A \mid \{ \overline{a} \langle V \rangle \}_B \xrightarrow{\text{Aerr}} \text{ if } c \notin B \]

Typable Processes do not violate privilege

- Stack Inspection Mechanism (JMV, SPIN, CLR)
- History-Based Mechanism [AF02]
Conclusion

- A New Expressive Theory of Types for the Higher-Order Code Mobility
- Applications to Secure Information Flow and Role-Based Access Control

Future Work

- Developing Secure Typing Systems for Programming Languages
  JFlow Multi-threaded Dynamic Class / Code Loading (Serialization)

Reference

- Full Version www.doc.ic.ac.uk/~yoshida
- Safe Dpi with Hennessy / Rathke [FoSSaCS04]
Script Server

$FW = \lambda (x: \mathbb{Z}). \lambda (y: \mathbb{Z}). x (z). y (z)$

forwarder

C1 evolves into

$FW(ab)$

$\begin{array}{l}
  : [a: (\mathbb{Z})^1, b: (\mathbb{Z})^0] \\
\end{array}$

C2 evolves into

$FW(ba)$

$\begin{array}{l}
  : [a: (\mathbb{Z})^0, b: (\mathbb{Z})^1] \\
\end{array}$

$(\mathbb{Z})^1 \rightarrow (\mathbb{Z})^0 \rightarrow \text{proc C}$

Func Dep [YHOO]

$\prod (x: (\mathbb{Z})^1) \prod (y: (\mathbb{Z})^0) [x: (\mathbb{Z})^1, y: (\mathbb{Z})^0]$