A Simple View of Type-Secure Information Flow in the $\pi$-Calculus
Outline

1. Type-based information flow analysis
2. A modular proof technique
3. Illustration 1: $\pi$-calculus under may-testing equivalence
4. Illustration 2: $\pi$-calculus under weak bisimulation equivalence (this paper)
5. Conclusion
Type-based information flow analysis
Defining information flow

Consider a sequential program $P$ of one input and one output. $P$ allows (some) information to flow from its input to its output if varying the former causes the latter to vary, that is, if the latter depends on the former:

$$\exists xy \quad P(x) \neq P(y)$$

The negation is called non-interference (Goguen and Meseguer, 1982):

$$\forall xy \quad P(x) = P(y)$$

More generally, if $P$ is a process with one input and $\approx$ is a notion of process equivalence, then

$$\forall xy \quad P(x) \approx P(y)$$

states that there is no flow of information from $P$’s input to “the observer”.

Type-based information flow analysis
Language-based (type-based) information flow control

- The system is (viewed as) a *program*,
- whose *semantics* it is easy to reason about,
- making a *static* analysis possible.

- Type-based analyses are *compositional*.
- Types can be viewed as a *specification* language.

- This approach yields only *typed* observational equivalences.
A modular proof technique
A modular proof technique

A type-based information flow analysis can be proved correct as follows:

- define an instrumented semantics, that is, a *dynamic* dependency analysis;
- prove it correct;
- define a type system for it, that is, a *static* approximation of it;
- prove it correct (*subject reduction*);
- derive a non-interference statement from the above.

The first part is about making dependencies explicit, and need not be concerned with types. The second part is a standard type preservation argument (albeit for a non-standard semantics).
Illustration 1: $\pi$-calculus under may-testing equivalence
Dynamic analysis: a labelled $\pi$-calculus

A $\pi$-calculus where (say) messages are labelled ($\ell \in \{L, H\}$):

$$ P ::= 0 \mid (P \mid P) \mid \nu x.P \mid !P \mid x(\tilde{y}).P \mid \ell : \tilde{x}\langle\tilde{z}\rangle $$

Operational semantics:

$$ x(\tilde{y}).P \mid \ell : \tilde{x}\langle\tilde{z}\rangle \rightarrow \ell \bullet P[\tilde{z}/\tilde{y}] $$


Illustration 1: $\pi$-calculus under may-testing equivalence
Properties of the labelled $\pi$-calculus

A prefix ordering is generated by $0 \leq P$. An erasure function is generated by $[H : \bar{x} \langle \bar{z} \rangle] = 0$. Then:

*Adding more sub-processes does not prohibit existing reductions.*

**Monotonicity.** If $P \rightarrow Q$ and $P \leq P'$, then $P' \rightarrow \cdot \geq Q$.

*Reducts of high-level sub-processes are high-level.*

**Stability (1).** If $P \rightarrow Q$, then $[P] \rightarrow \cdot \geq [Q]$.

Corollary:

**Stability (*).** If $P \rightarrow^* Q$, then $[P] \rightarrow^* \cdot \geq [Q]$.

Illustration 1: $\pi$-calculus under may-testing equivalence
Static approximation: typing the labelled $\pi$-calculus

Let types be given by $t ::= \langle \tilde{t} \rangle^\ell$. Define a type system which satisfies the following properties:

*Types are preserved by reduction.*

**Subject reduction.** $P \rightarrow Q$ and $\Gamma \vdash P$ imply $\Gamma \vdash Q$.

*Messages on channels of low type have low labels.*

**Barb preservation.** If $x : \langle \rangle^L \vdash P$ and $P \Downarrow x$, then $\llbracket P \rrbracket \Downarrow x$.

Note that this type system guarantees a *safety* property. Its design is guided by the labelled semantics.

Illustration 1: $\pi$-calculus under may-testing equivalence
Non-interference statements

Weak barbs on channels of low type are preserved by erasure.

**Non-interference.** If $x : \langle \rangle^L \vdash P$ and $P \downarrow_x$, then $\lfloor P \rfloor \downarrow_x$.

**Proof.** Assume $P \rightarrow^* P'$ and $P' \downarrow_x$. By subject reduction and barb preservation, $\lfloor P' \rfloor \downarrow_x$. Furthermore, by stability, $\lfloor P \rfloor \rightarrow^* \geq \lfloor P' \rfloor$. This implies $\lfloor P \rfloor \downarrow_x$.

Two processes which differ only in high-level components have the same weak barbs on channels of low type.

**Non-interference.** If $x : \langle \rangle^L \vdash P, Q$ and $\lfloor P \rfloor = \lfloor Q \rfloor$ then $P \approx_{may} Q$.

Illustration 1: $\pi$-calculus under may-testing equivalence
Illustration 2: $\pi$-calculus under weak bisimulation equivalence
May-testing vs. weak bisimulation equivalence

The process $\nu y. (y.\bar{x} \mid \bar{y} \mid H : y.0)$ only has low barbs at $x$, so it is may-testing equivalent to its erasure $\nu y. (y.\bar{x} \mid \bar{y})$. Yet they are not bisimilar, since the former may remain silent forever, while the latter must emit a signal on $x$.

Thus, under bisimulation equivalence, information may flow between several receivers on a single channel.

The dynamic dependency analysis, as well as its static counterpart, must then report more potential dependencies.
Dynamic analysis: the $\langle \pi \rangle$-calculus

The $\langle \pi \rangle$-calculus is defined as an extension of the $\pi$-calculus. (Brackets cannot be nested.)

$$P ::= \ldots \mid \langle P \rangle_1 \mid \langle P \rangle_2$$

A $\langle \pi \rangle$-calculus term encodes a \textit{pair} of $\pi$-calculus terms. For instance, $P \mid \langle Q \rangle_1$ and $\langle P \mid Q \rangle_1 \mid \langle P \rangle_2$ both encode the pair $(P \mid Q, P)$.

Brackets encode the \textit{differences} between two processes, i.e. their high-level parts, while the low-level parts are \textit{shared}.

Two \textit{projection} functions map a $\langle \pi \rangle$-calculus term to the two $\pi$-calculus terms which it encodes. In particular, $\lfloor \langle P \rangle_i \rfloor_i = P$ and $\lfloor \langle P \rangle_j \rfloor_i = 0$, for \{i, j\} = \{1, 2\}.

Inspired by joint work with Vincent Simonet (POPL 2002).
Semantics

Communication is dealt with by *two* reduction rules: a standard one, and one that moves brackets out of the way.

\[ x(\tilde{y}).P \mid \tilde{x}\langle \tilde{z} \rangle \rightarrow P[\tilde{z}/\tilde{y}] \]

\[ M \mid \langle N \rangle_i \rightarrow \langle [M]_i \mid N \rangle_i \mid \langle [M]_j \rangle_j \]

if \( \{i, j\} = \{1, 2\} \)

and \([M]_i \mid N\) may react

The former applies within or outside brackets. The latter leaves both projections unchanged; it only keeps track of dependencies. Note that it reflects the flow of information even in the *absence* of communication.

Illustration 2: \( \pi \)-calculus under weak bisimulation equivalence
Properties of the $\langle \pi \rangle$-calculus

The $\langle \pi \rangle$-calculus encodes valid reductions only.

**Soundness.** If $P \rightarrow P'$, then $\lfloor P \rfloor_i \rightarrow^* \lfloor P' \rfloor_i$.

The $\langle \pi \rangle$-calculus encodes all valid reductions.

**Completeness.** (simplified) Assume $\lfloor P \rfloor_i \rightarrow Q$. Then, there exists $P'$ such that $P \rightarrow^* P'$ and $\lfloor P' \rfloor_i = Q$.

In short, projection establishes a (weak) *bisimulation* between the $\pi$-calculus and the $\langle \pi \rangle$-calculus.

Illustration 2: $\pi$-calculus under weak bisimulation equivalence
Static approximation: typing the $\langle \pi \rangle$-calculus

Let types be given by $t ::= \langle \tilde{t} \rangle^\ell$. Define a type system which satisfies the following properties:

*Types are preserved by reduction.*

*Subject reduction.* $P \rightarrow Q$ and $\Gamma \vdash P$ imply $\Gamma \vdash Q$.

*Messages on channels of low type cannot appear within brackets.*

*Barb preservation.* If $x : \langle \rangle^L \vdash P$ and $P \downarrow_x$, then $\left\lfloor P \right\rfloor \downarrow_x$.

Again, this type system guarantees a *safety* property. Again, its design is guided by the semantics of the $\langle \pi \rangle$-calculus.

Illustration 2: $\pi$-calculus under weak bisimulation equivalence
Non-interference statement

**Non-interference.** If $x : \langle \rangle^L \vdash P$, then $\lfloor P \rfloor_1 \approx \lfloor P \rfloor_2$.

One may say that the $\langle \pi \rangle$-calculus and its type system are simply a structured description of the bisimulation *invariant*.

Illustration 2: $\pi$-calculus under weak bisimulation equivalence
Conclusion
Conclusion

I have sketched a couple of *two-step* approaches to establishing the correctness of a type-based information flow analysis, separating a purely *dynamic* analysis, on the one hand, and a *static* approximation, on the other hand.

- these approaches yield manageable, modular proofs;
- on the down-side, not all analyses can be decomposed in this way. For instance, the dynamic analysis may require type information, introducing a circularity.