Information Flow Security for Concurrent Programs

A. Bossi, C. Piazza, and S. Rossi

University Ca’ Foscari of Venezia
University of Udine

\{bossi,piazza,srossi\}@dsi.unive.it

CSFW 2006, Luglio 2006, Venezia
Security as Unwinding - Intuition

If the high level user can perform $h$ reaching $E''$ from $E'$, then also $E'''$ is reachable from $E'$ and $E''$ and $E'''$ are low level undistinguishable.

Depends on: operational semantics, reachability, undistinguishability
Basic Concurrent Imperative Language

\[
P ::= \quad \text{skip} \quad \text{skip}
\]
\[
| \quad X := a \quad \text{assignement}
\]
\[
| \quad P; P \quad \text{sequence}
\]
\[
| \quad \text{if}(b) \text{ then } P \text{ else } P \quad \text{conditional}
\]
\[
| \quad \text{while}(b) \text{ do } P \quad \text{loop}
\]
\[
| \quad \text{await}(b)\{P\} \quad \text{atomic}
\]
\[
| \quad \text{co } P \| \ldots \| P \text{ oc} \quad \text{parallel}
\]

Toghehte with: standard operational semantics, bisimulation, trace equivalence
Unwininding conditions are usually undecidable over Concurrent Imperative Languages since:

▷ there is an infinite number of states

▷ integer numbers are involved (10th Hilbert problem)

We define a correct verification method combining static and dynamic analysis:

▷ we define a proof system

▷ we exploit symbolic algorithms over the reals whenever it is possible
Example

Consider the program $P \equiv \text{co } P_0 \parallel P_1 \text{ oc}$ where $P_0$ and $P_1$ are defined as

$$P_i \equiv \text{while(true) } \{
\text{await}(M = i) \{
L := f_i(H_1, H_2);
H_1 := g_i(H_1, H_2);
H_2 := L;
L := 0;
M := (i + 1) \text{ mod } 2
\}\}
\}$$

where $M$ is a low level variable ensuring mutual exclusion.

We can automatically prove that $P$ is secure.
So what?

The main **advantages** of the framework are:

- flexibility, e.g., *downgrading* can be easily embedded
- higher precision can be achieved

The main **issues** are:

- how to further increment the precision extending the symbolic dynamic techniques to recursive programs?
- how to reduce the complexity of the method?
- how to extend the concurrent imperative language?