

Information Flow Security for Concurrent Programs

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

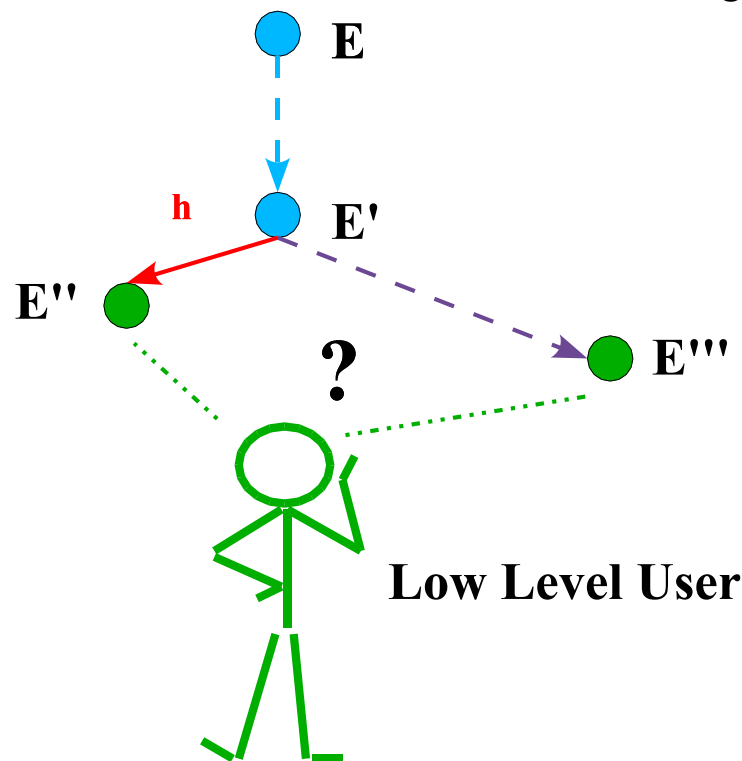
**University Ca' Foscari of Venezia
University of Udine**

{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	$::=$	<code>skip</code>	<i>skip</i>
		<code>$X := a$</code>	<i>assignement</i>
		<code>$P; P$</code>	<i>sequence</i>
		<code>if(b) then P else P</code>	<i>conditional</i>
		<code>while(b) do P</code>	<i>loop</i>
		<code>await(b){P}</code>	<i>atomic</i>
		<code>co P ... P oc</code>	<i>parallel</i>

Togheter with: **standard operational semantics, bisimulation, trace equivalence**

Verification Techniques

Unwinding 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

$$\begin{aligned} P_i \equiv & \text{while}(\text{true}) \{ \\ & \text{await}(M = i) \{ \\ & \quad L := f_i(H_1, H_2); \\ & \quad H_1 := g_i(H_1, H_2); \\ & \quad H_2 := L; \\ & \quad L := 0; \\ & \quad M := (i + 1) \bmod 2 \\ & \} \\ & \} \end{aligned}$$

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?