Information Flow Security for Concurrent Programs

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

University Ca' Foscari of Venezia

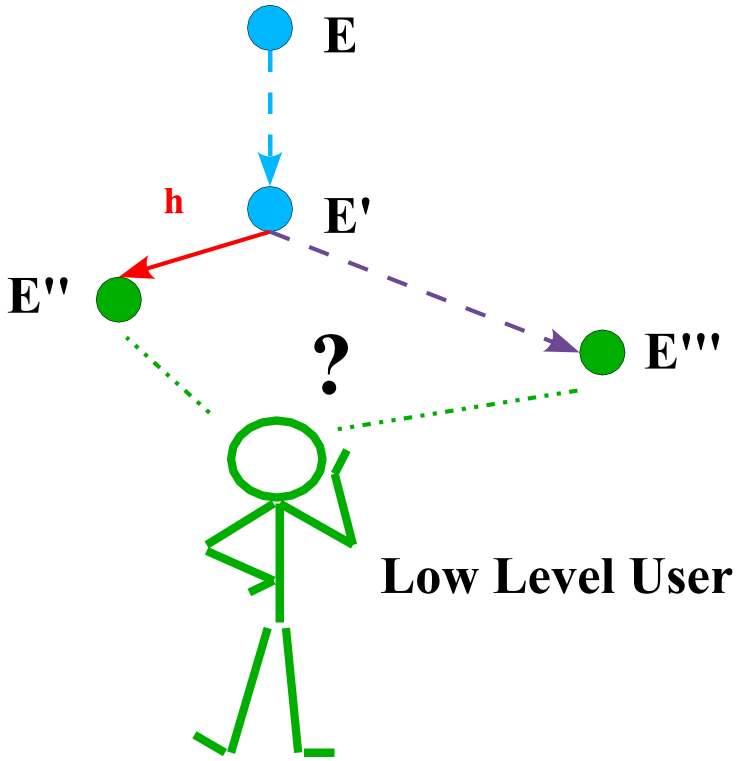
University of Udine

{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	$::=$	$\texttt{skip}$	*skip*
	$\mid$	$X := a$	*assignement*
	$\mid$	$P; P$	*sequence*
	$\mid$	$\texttt{if}(b)\ \texttt{then}\ P\ \texttt{else}\ P$	*conditional*
	$\mid$	$\texttt{while}(b)\ \texttt{do}\ P$	*loop*
	$\mid$	$\texttt{await}(b)\{P\}$	*atomic*
	$\mid$	$\texttt{co}\ P \| \ldots \| P\ \texttt{oc}$	*parallel*

Togheter with: standard operational semantics, bisimulation, trace equivalence

Verification Techniques

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 $P \equiv \texttt{co}\ P_0 \| P_1\ \texttt{oc}$ where P_0 and P_1 are defined as

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

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?