

An Unwinding Condition for Security in Imperative Languages

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

**Department of Computer Science
University Ca' Foscari of Venezia**

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

LOPSTR 2004, August 23-28, 2004, Verona

Protect Confidential Data in a Multilevel System

- ▷ **Information Flow Security** aims at guaranteeing that no **high** level (**confidential**) information is revealed to users at **low** level, even in the presence of any possible **malicious process**
- ▷ **Non-Interference** [Goguen-Meseguer'82]: **information does not flow** from high to low if the **high behavior** has **no effect** on what **low** level can **observe**
- ▷ **Development of Complex Systems**: it is important to **preserve the security** properties of interest during the **development steps**

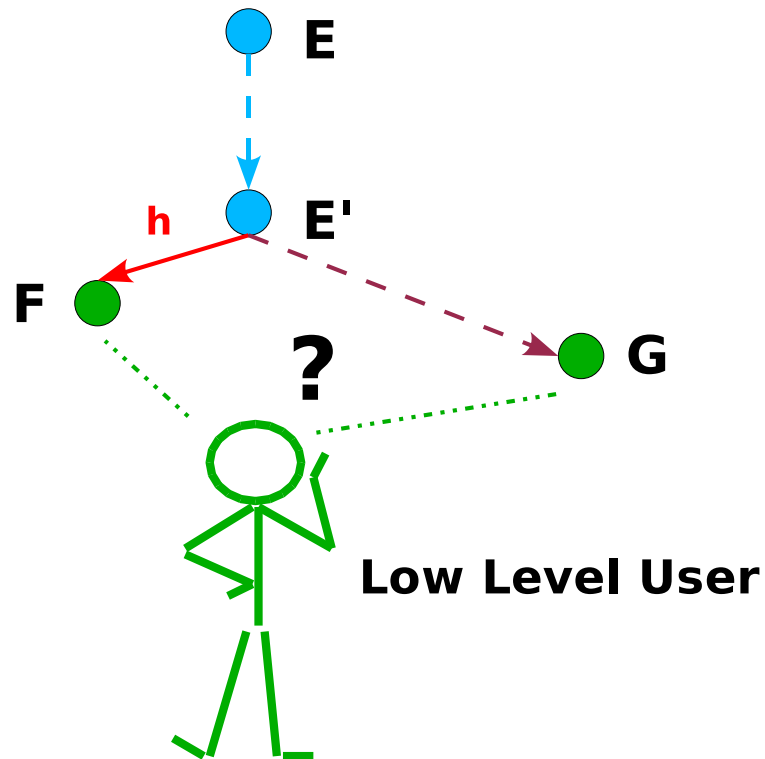
Active and Passive attacks

- ▷ **Active Attacks:** a **high level malicious** process sends down **high level information** to the **low level** user
difficult to check - trojan horse
- ▷ **Passive Attacks:** the **low level** user observing the execution infers the **high level behaviour**
easy to check - unwinding condition

Active Attacks \Rightarrow Passive Attacks

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 undistinguishable for the **low** level user



In **process algebra** many **security properties** are based on this **schema**

Plan of the Talk

- ▷ The **IMP** language: syntax and semantics
- ▷ The **SIMP Unwinding class**
- ▷ **Characterization** and **Properties**
- ▷ A class of **Secure Programs**
- ▷ **Conclusions**

The IMP syntax

The locations X are partitioned into \mathbb{L} and \mathbb{H}

The set **Aexp** of arithmetic expressions is:

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1$$

The set **Bexp** of boolean expressions is:

$$b ::= \text{true} \mid \text{false} \mid (a_0 = a_1) \mid (a_0 \leq a_1) \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$

The set **Prog** of programs of our language is:

$$P ::= \text{skip} \mid X := a \mid P_0; P_1 \mid \text{if } b \text{ then } P_0 \text{ else } P_1 \mid \\ \text{while } b \text{ do } P \mid P_0 \parallel P_1$$

The IMP semantics - I

A state σ is a map from locations to values

An **expression** containing high level variables is **high level**

Semantics is given through **labelled transition systems (LTS)**:

- ▷ the **nodes** of the LTS are pairs $\langle P, \sigma \rangle$ of programs and states
- ▷ the **edges** are **labelled** with a security level, i.e., **high** or **low**

$$\frac{\langle a, \sigma \rangle \rightarrow n}{\langle X := a, \sigma \rangle \xrightarrow{\epsilon} \langle \text{end}, \sigma[X/n] \rangle} \quad a \in \epsilon$$

The IMP semantics - II

$$\frac{\langle P_0, \sigma \rangle \xrightarrow{\epsilon} \langle P'_0, \sigma' \rangle}{\langle P_0; P_1, \sigma \rangle \xrightarrow{\epsilon} \langle P'_0; P_1, \sigma' \rangle} \quad P'_0 \neq \text{end} \qquad \frac{\langle P_0, \sigma \rangle \xrightarrow{\epsilon} \langle \text{end}, \sigma' \rangle}{\langle P_0; P_1, \sigma \rangle \xrightarrow{\epsilon} \langle P_1, \sigma' \rangle}$$

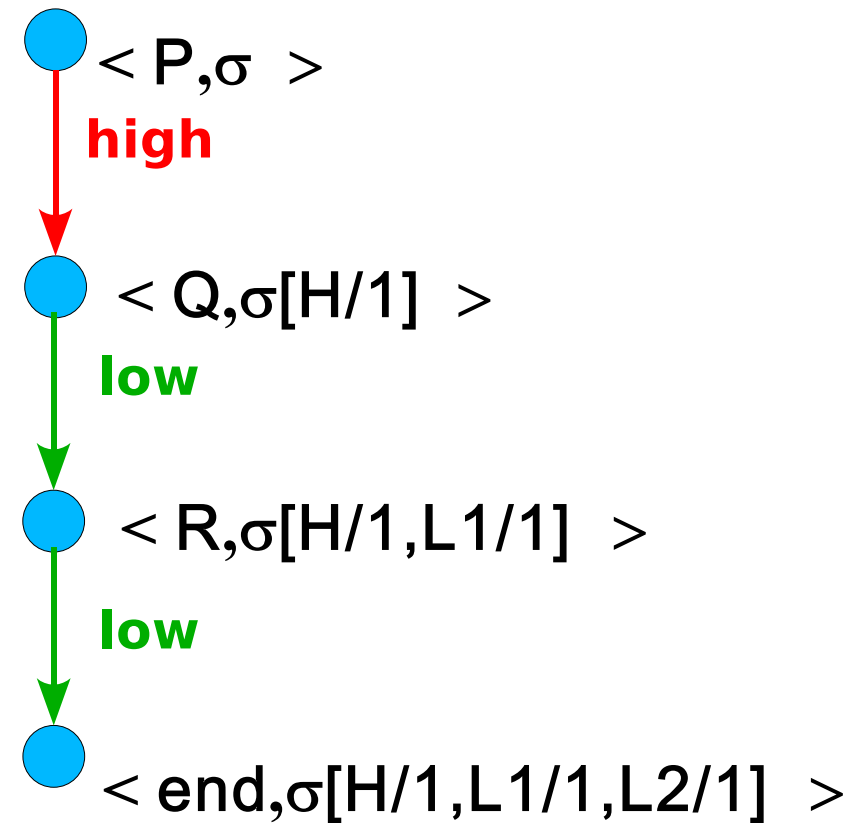
$$\frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{while } b \text{ do } P, \sigma \rangle \xrightarrow{\epsilon} \langle P; \text{while } b \text{ do } P, \sigma \rangle} \quad b \in \epsilon$$

$$\frac{\langle P_0, \sigma \rangle \xrightarrow{\epsilon} \langle P'_0, \sigma' \rangle}{\langle P_0 \parallel P_1, \sigma \rangle \xrightarrow{\epsilon} \langle P'_0 \parallel P_1, \sigma' \rangle} \quad P'_0 \neq \text{end} \qquad \frac{\langle P_0, \sigma \rangle \xrightarrow{\epsilon} \langle \text{end}, \sigma' \rangle}{\langle P_0 \parallel P_1, \sigma \rangle \xrightarrow{\epsilon} \langle P_1, \sigma' \rangle}$$

Behavioral equivalences establish equalities among programs

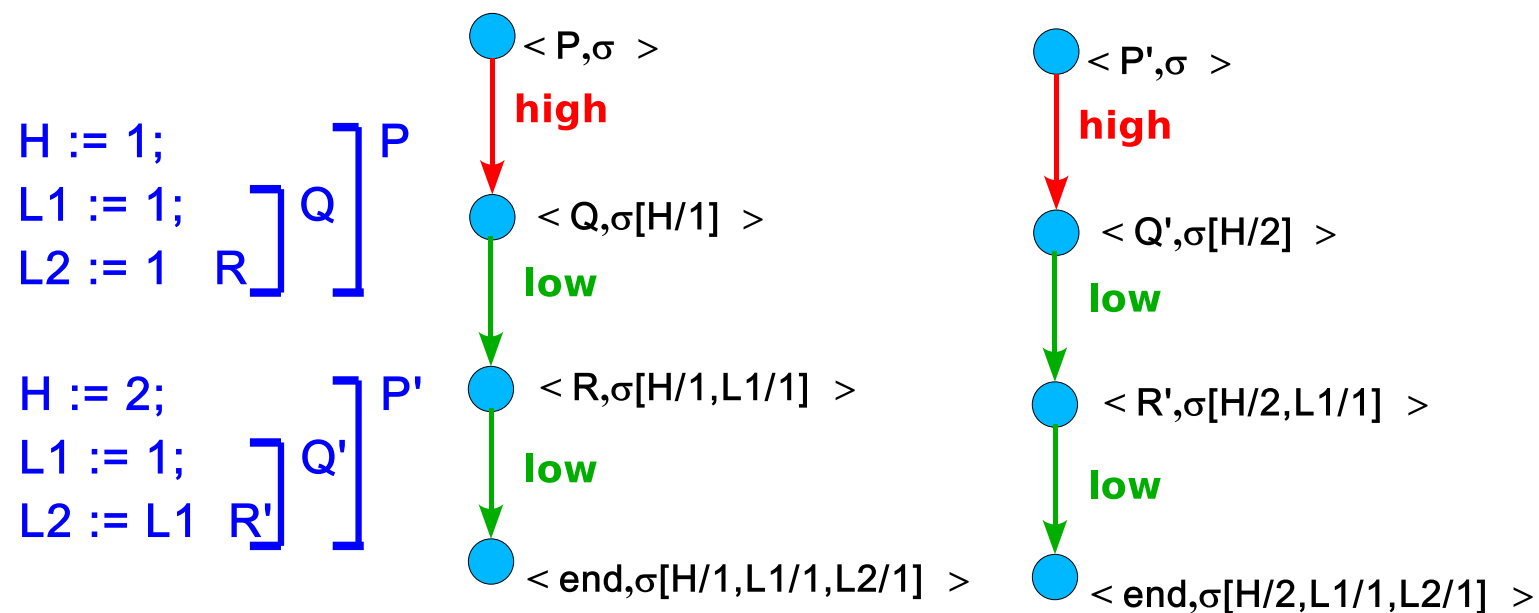
Example

H := 1;
L1 := 1; R] Q] P
L2 := 1 R] Q] P



Low Level Bisimulation

Idea: it is a mutual step-by-step simulation on low level locations



$$\langle P, \sigma \rangle \sim_l \langle P', \sigma \rangle \quad \text{and} \quad (\sigma[H/1])_L = (\sigma[H/2])_L$$

The Unwinding Condition

$\langle F, \psi \rangle \dashrightarrow \langle M, \mu \rangle$ iff for all π , such that $\pi =_L \psi$, there exist R and ρ such that $\langle F, \pi \rangle \rightarrow \langle R, \rho \rangle$ and $\langle R, \rho \rangle \sim_l \langle M, \mu \rangle$

$LReach(\langle P_0, \sigma_0 \rangle) = \{ \langle P_n, \theta_n \rangle \mid \text{there exist } n \geq 0, P_0, \dots, P_n, \sigma_0, \dots, \sigma_n, \theta_0, \dots, \theta_n \text{ such that } \sigma_i =_L \theta_i \text{ and } \langle P_i, \theta_i \rangle \rightarrow \langle P_{i+1}, \sigma_{i+1} \rangle, 0 \leq i \leq n \}$

$\mathcal{W}(\sim_l, \dashrightarrow, LReach) \stackrel{\text{def}}{=} \{ \langle P, \sigma \rangle \in \mathbf{Prog} \times \Sigma \mid \forall \langle F, \psi \rangle \in LReach(\langle P, \sigma \rangle) \text{ if } \langle F, \psi \rangle \xrightarrow{\text{high}} \langle G, \varphi \rangle \text{ then } \exists \langle M, \mu \rangle \text{ such that } \langle F, \psi \rangle \dashrightarrow \langle M, \mu \rangle \text{ and } \langle G, \varphi \rangle \sim_l \langle M, \mu \rangle \}$

Characterization and Soundness

P is in $\mathcal{W}(\sim_L, \dashrightarrow, LReach)$ iff

$\langle F, \psi \rangle \in LReach(\langle P, \sigma \rangle)$ and $\langle F, \psi \rangle \xrightarrow{high} \langle G, \varphi \rangle$ imply that for each π such that $\pi_L = \psi_L$ there exist R and ρ such that $\langle F, \pi \rangle \rightarrow \langle R, \rho \rangle$ and $\langle R, \rho \rangle \sim_L \langle G, \varphi \rangle$

If P is in $\mathcal{W}(\sim_L, \dashrightarrow, LReach)$ then

for each σ and θ such that $\sigma_L = \theta_L$, if $\langle P, \sigma \rangle$ reaches $\langle \text{end}, \sigma' \rangle$, then $\langle P, \theta \rangle$ reaches $\langle \text{end}, \theta' \rangle$ with $\sigma'_L = \theta'_L$

Examples

$P \equiv \text{if } (L = 1) \text{ then } H := H + 1 \text{ else } L := L + 1$

is secure

$R \equiv H := 4; L := 1; \text{if } (L = 1) \text{ then skip else } L := H$

is secure

$S \equiv L := 4$

is secure

BUT $R \parallel S$ is not secure

A Class of Secure Programs

Let H be a high level location, L be a low level location, a_h and b_h be high level expressions, and a_l and b_l be low level expressions. The class of programs \mathcal{C} is recursively defined as follows.

1. skip is in \mathcal{C} ;
2. $L := a_l$, $H := a_h$, and $H := a_l$ are in \mathcal{C} ;
3. $P_0; P_1$ is in \mathcal{C} if P_0, P_1 are in \mathcal{C} ;
4. $\text{if } b_l \text{ then } P_0 \text{ else } P_1$ is in \mathcal{C} , if P_0, P_1 are in \mathcal{C} ;
5. $\text{if } b_h \text{ then } P_0 \text{ else } P_1$ is in \mathcal{C} if P_0, P_1 are in \mathcal{C} and $P_0 \sim_l P_1$;
6. $\text{while } b_l \text{ do } P_0$ is in \mathcal{C} , if P_0 is in \mathcal{C} ;
7. $P_0 \parallel P_1$ is in \mathcal{C} , if P_0, P_1 are in \mathcal{C} .

Conclusions

- ▷ We introduce an **unwinding condition** which ensures the absence of **passive attacks** in the context of an **imperative concurrent language**
- ▷ We provide a **syntactic characterization** of a class of **secure programs**
- ▷ We are studying **semantics** decision procedures to **check** security
- ▷ We are extending the framework to relax non-interference condition (downgrading)