Typing Noninterference for a Reactive Language

Ana Almeida Matos

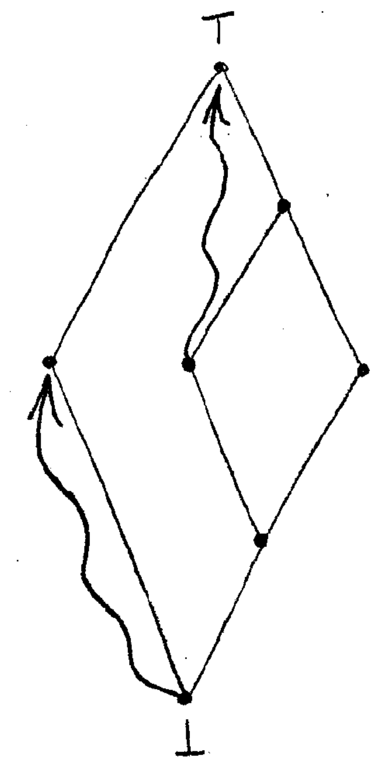
joint work with Gérard Boudol and Ilaria Castellani

INRIA Sophia Antipolis

Topics

- **Noninterference for an imperative language**
  - **Security leaks introduced by concurrency**
  - **Typing rules that prevent them**
- Reactive Languages
- Noninterference for Reactive Languages
- Proving Noninterference

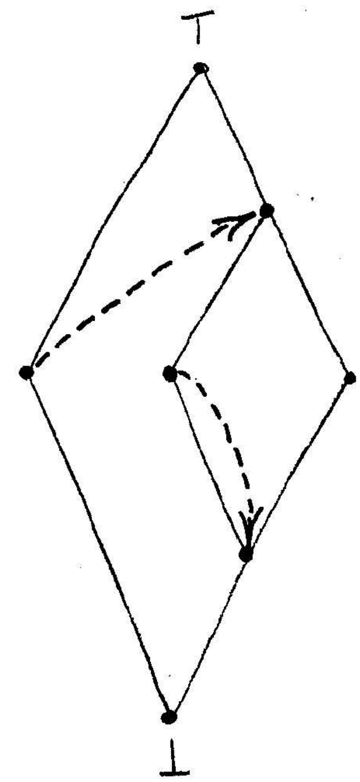
Noninterference property



No variable is **ever**
influenced by
higher or incomparable
level variables.

Simplification to the
lattice model:

$$H$$
$$|$$
$$L$$



Examples – Interference

Direct flow

$$y_L := x_H$$ ⛔

Indirect flow

$$\texttt{if}\ x_H\ \texttt{then}\ y_L := 0\ \texttt{else}\ y_L := 1$$ ⛔

$$y_L := 1\,;\ \texttt{while}\ x_H\ \texttt{do}\ (y_L := 0\,;\ x_H := \texttt{false})$$ ⛔

What is wrong? Low variables are assigned under high tests!

The context matters

Sequential language :

$$\texttt{while}\ x_H\ \texttt{do nil}\,;\, r_L := 0$$

Concurrent language :

$$(\alpha|\beta)$$

$$\alpha : \texttt{while}\ x_H\ \texttt{do nil}\,;\, r_L := 0\,;\, x_H := \texttt{false}$$

$$\beta : \texttt{while}\ \neg x_H\ \texttt{do nil}\,;\, r_L := 1\,;\, x_H := \texttt{true}$$

$$\{x_H \mapsto \texttt{true}\}\ \text{vs.}\ \{x_H \mapsto \texttt{false}\}$$

Rationale behind the types

Following [BouCas01,02] and [Smith01].

$$(P_1 \quad ; \quad P_2)$$

$$\swarrow \qquad\qquad \searrow$$

$$\text{highest-tests}(P_1) \quad \leq \quad \text{lowest-writes}(P_2)$$

What a type must tell about programs:

- an upper bound to test-levels
- a lower bound to write-levels

Giving (double) types

	Statement	Property
Variables	$\Gamma(x) = \sigma\ var$	$\sigma =$ the security level of x
Expressions	$\Gamma \vdash e : \sigma$	$\sigma \geq$ level of read variables
Commands	$\Gamma \vdash P : (\tau, \sigma)\ cmd$	$\tau \leq$ level of written variables
		$\sigma \geq$ level of read variables

Subtyping:
$$\frac{\Gamma \vdash P : (\tau, \sigma)\ cmd \quad \tau \geq \tau' \quad \sigma \leq \sigma'}{\Gamma \vdash P : (\tau', \sigma')\ cmd}$$

Typing rules - Imperative primitives

All writes after a read must be higher:

$$\frac{\Gamma \vdash e : \boldsymbol{\delta} \quad \Gamma \vdash P : (\tau, \sigma)\ cmd \quad \Gamma \vdash Q : (\tau, \sigma)\ cmd}{\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q : (\tau, \boldsymbol{\delta} \vee \boldsymbol{\sigma})\ cmd} \qquad \underbrace{\text{must be higher:}}\ \boldsymbol{\delta} \leq \boldsymbol{\tau}$$

$$\frac{\Gamma \vdash Q_1 : (\tau_1, \sigma_1)\ cmd \quad \Gamma \vdash Q_2 : (\tau_2, \sigma_2)\ cmd}{\Gamma \vdash Q_1\,;\, Q_2 : (\boldsymbol{\tau_1} \wedge \boldsymbol{\tau_2}, \boldsymbol{\sigma_1} \vee \boldsymbol{\sigma_2})\ cmd} \qquad \boldsymbol{\sigma_1} \leq \boldsymbol{\tau_2}$$

Topics

- Noninterference for an imperative language
- **Reactive Languages**
  - **Motivation and informal semantics**
  - **Examples**
- Noninterference for Reactive Languages
- Proving Noninterference

Reactive languages – Informal semantics

- $\texttt{emit}\ s$
- $\texttt{when}\ a\ \texttt{do}\ P$
- $\texttt{do}\ P\ \texttt{watching}\ a$
- $P \Lsh Q$

Environment (E) Set of emitted signals.

Suspension $(\dagger)$ Pendency due to absent signals.

Instant Interval in which signals are present or absent.

$$\underbrace{\langle \emptyset, P\rangle \longrightarrow^* \langle E_1, P_1\rangle\dagger}_{\text{Instant 1}} \quad \hookrightarrow \quad \underbrace{\langle \emptyset, P_1'\rangle \longrightarrow^* \langle E_2, P_2\rangle\dagger}_{\text{Instant 2}} \quad \hookrightarrow \quad \ldots$$

Example – Instant change ($\hookrightarrow$)

$$\frac{\langle E, P\rangle\dagger}{\langle E, P\rangle \hookrightarrow \langle\emptyset, \quad \lfloor P\rfloor_E\rangle}$$

$\swarrow$ initialize E $\qquad$ $\searrow$ perform kills

Environment	**"pause"**
$\{\}$	$\texttt{local } a : \delta \texttt{ in } (\texttt{emit } b \texttt{; do when } a \texttt{ do nil watching } b)$
$\rightarrow$ $\{b\}$	$\underbrace{\texttt{do } \underbrace{\texttt{when } a' \texttt{ do nil}}_{\dagger} \texttt{ watching } b}_{\dagger}$
$\hookrightarrow$ $\{\}$	$\texttt{nil}$

Example – Deterministic concurrency $\Lsh$

	Environment	Watch the position of the threads:
	$\{\}$	$(\underbrace{\texttt{when}\ a\ \texttt{do emit}\ b \Lsh \texttt{when}\ b\ \texttt{do emit}\ c}_{\dagger}) \Lsh \texttt{emit}\ a$
$\longrightarrow$	$\{\}$	$\texttt{emit}\ a \Lsh (\texttt{when}\ a\ \texttt{do emit}\ b \Lsh \texttt{when}\ b\ \texttt{do emit}\ c)$
$\longrightarrow$	$\{a\}$	$\texttt{when}\ a\ \texttt{do emit}\ b \Lsh \texttt{when}\ b\ \texttt{do emit}\ c$
$\longrightarrow$	$\{a, b\}$	$\texttt{when}\ b\ \texttt{do emit}\ c$
$\longrightarrow$	$\{a, b, c\}$	$\texttt{nil}$

Topics

- Noninterference for an imperative language
- Reactive Languages
- **Noninterference for Reactive Languages**
  - **Some old and new security leaks**
  - **Typing rules that prevent them**
- Proving Noninterference

Reactive noninterference: what is the right notion?

Also here we can use double types.

Should we allow ...?

- `while` x_H `do` $\ldots; \ldots r_L := 0 \ldots$
- `when` a_H `do` $\ldots\ ;\ \ldots r_L := 0 \ldots$
- `when` a_H `do` $\ldots \Lsh \ldots r_L := 0 \ldots$

Counter-example I

$$\texttt{while}\ x_H\ \texttt{do}\ \ldots ; \ldots r_L := 0 \ldots$$

because:

$$(\alpha \Lsh \beta)$$

$$\alpha : \texttt{while}\ x_H\ \texttt{do pause}\,;\, r_L := 0\,;\, x_H := \texttt{false}$$

$$\beta : \texttt{while}\ \neg x_H\ \texttt{do pause}\,;\, r_L := 1\,;\, x_H := \texttt{true}$$

$$\{x_H \mapsto \texttt{true}\}\ \text{vs.}\ \{x_H \mapsto \texttt{false}\}$$

Counter-example II

$$\texttt{when } a_H \texttt{ do } \ldots ; \ldots r_L := 0 \ldots$$

because:

$$(\alpha \Lsh \beta)$$

$$\alpha : \texttt{when } a_H \texttt{ do nil} ; \texttt{emit } c_L ; \texttt{emit } b_H$$

$$\beta : \texttt{when } b_H \texttt{ do nil} ; \texttt{emit } d_L ; \texttt{emit } a_H$$

$$\{a_H\} \text{ vs. } \{b_H\}$$

Counter-example III

$$\texttt{when } a_H \texttt{ do } \ldots \Lsh \ldots r_L := 0 \ldots$$

because:

$$((\alpha \Lsh \beta) \Lsh \gamma)$$

$$\alpha : (\texttt{pause} \,;\, x_L := 1)$$

$$\beta : (\texttt{do } (\texttt{when } a_H \texttt{ do nil}) \texttt{ watching } z_H \Lsh \texttt{when } b_L \texttt{ do } x_L := 0)$$

$$\gamma : (\texttt{nil} \,;\, \texttt{pause} \,;\, \texttt{emit } b_L)$$

$$\{a_H, z_H\} \text{ vs. } \{z_H\}$$

$\therefore$ Alternating parallelism requires more conditions

Double types for reactive primitives

All writes after a read must be higher:

$\checkmark$ $$\frac{\Gamma(a) = \boldsymbol{\delta}\ sig \quad \Gamma \vdash P : (\tau, \sigma)\ cmd}{\Gamma \vdash \texttt{when}\ a\ \texttt{do}\ P : (\tau, \boldsymbol{\delta} \vee \boldsymbol{\sigma})\ cmd} \qquad \boldsymbol{\delta} \leq \boldsymbol{\tau}$$

$\checkmark$ $$\frac{\Gamma \vdash Q_1 : (\tau_1, \sigma_1)\ cmd \quad \Gamma \vdash Q_2 : (\tau_2, \sigma_2)\ cmd}{\Gamma \vdash Q_1\ ;\ Q_2 : (\boldsymbol{\tau_1} \wedge \boldsymbol{\tau_2}, \boldsymbol{\sigma_1} \vee \boldsymbol{\sigma_2})\ cmd} \qquad \boldsymbol{\sigma_1} \leq \boldsymbol{\tau_2}$$

! $$\frac{\Gamma \vdash Q_1 : (\tau_1, \sigma_1)\ cmd \quad \Gamma \vdash Q_2 : (\tau_2, \sigma_2)\ cmd}{\Gamma \vdash Q_1 \urcorner Q_2 : (\boldsymbol{\tau_1} \wedge \boldsymbol{\tau_2}, \boldsymbol{\sigma_1} \vee \boldsymbol{\sigma_2})\ cmd} \qquad \boldsymbol{\sigma_1} \leq \boldsymbol{\tau_2} \wedge \boldsymbol{\sigma_2} \leq \boldsymbol{\tau_1}$$

Interference at instant changes

Instant changes...

...are reflected in the signal environment (it is set to empty).

...can depend on high tests

$$\texttt{emit}\ a_L\ ;\ \texttt{if}\ x_H = 0\ \texttt{then nil else pause}$$

...are not statically predictable

$\therefore$ So we **must** allow **some** low signal reset after a high test.

Topics

- Noninterference for an imperative language
- Reactive Languages
- Noninterference for Reactive Languages
- **Proving Noninterference**
  - **The language and some properties**
  - **Noninterference using bisimulation**
  - **High programs**

The language

Imperative

$\texttt{nil} \mid x := e \mid \texttt{let } x := e \texttt{ in } P \mid \texttt{if } e \texttt{ then } P \texttt{ else } Q \mid \texttt{while } e \texttt{ do } P \mid P\,;Q$

Reactive

$\texttt{emit } a \mid \texttt{local } a : \delta \texttt{ in } P \mid \texttt{do } P \texttt{ watching } a \mid \texttt{when } a \texttt{ do } P \mid P \Lsh Q$

Configuration $C_1, C_2, \ldots = \langle \Gamma, S, E, P \rangle$ where:

Γ - typing environment $\quad S$ - variable store

E - set of present signals $\quad P, Q$ - programs

Step $C \longmapsto C'$ $\quad \stackrel{\text{def}}{\Leftrightarrow} \quad$ **Move** $C \rightarrow C'$ or **Instant change** $C \hookrightarrow C'$.

Formalizing Noninterference

Idea: a program should be bisimilar to itself when executed on low-equal memories (bisimulation preserving low memories):

Definition 1 (Secure Programs).
P is secure in Γ if for all set of low security levels $\mathcal{L}$ and for all S_1, E_1, S_2, E_2 such that $\langle S_1, E_1 \rangle =^{\Gamma}_{\mathcal{L}} \langle S_2, E_2 \rangle$, we have

$$\langle \Gamma, S_1, E_1, P \rangle \approx_{\mathcal{L}} \langle \Gamma, S_2, E_2, P \rangle.$$

Reactive bisimulation

Definition 2 (Reactive bisimulation equivalence ($\approx_{\mathcal{L}}$)). *The largest symmetric relation $\mathcal{R}$ such that $C_1 \mathcal{R} C_2$, where $C_1 = \langle \Gamma_1, S_1, E_1, P_1 \rangle$ and $C_2 = \langle \Gamma_2, S_2, E_2, P_2 \rangle$, imply:*

- $\langle S_1, E_1 \rangle =_{\mathcal{L}}^{\Gamma_1 \cap \Gamma_2} \langle S_2, E_2 \rangle$, *and*
- *either*
  - $P_i \in \mathcal{H}^{\Gamma_i, \mathcal{L}}$ *for* $i = 1, 2$, *or*
  - $C_1 \longmapsto C_1'$ *implies* $\exists C_2'$ *such that* $C_2 \longmapsto^{\star} C_2'$ *and* $C_1' \mathcal{R} C_2'$

Semantically High programs – $\mathcal{H}^{\Gamma,\mathcal{L}}$

Definition: $P \in \mathcal{H}^{\Gamma,\mathcal{L}}$ implies

- $\langle \Gamma, S, E, P \rangle \rightarrow \langle \Gamma', S', E', P' \rangle$ implies $P' \in \mathcal{H}^{\Gamma',\mathcal{L}}$ and $\langle S, E \rangle =^{\Gamma}_{\mathcal{L}} \langle S', E' \rangle$
- $\langle \Gamma, S, E, P \rangle \hookrightarrow \langle \Gamma', S', E', P' \rangle$ implies $P' \in \mathcal{H}^{\Gamma',\mathcal{L}}$

Examples:

- `if` $x_H = 0$ `then nil else pause`
- `if true then nil else` $y_L := 0$

Main results

Lemma 3. *Suppose* $C_1 =_{\mathcal{L}}^{\Gamma} C_2$, $C_1 \longmapsto C_1'$ *and* $C_2 \longmapsto C_2'$.

1. *If* P *has only low tests, then* $C_1' =_{\mathcal{L}}^{\Gamma} C_2'$.
2. *If* P *has high tests and* $C_1' \neq_L C_2'$, *then* $P \in \mathcal{H}^{\Gamma,\mathcal{L}}$.

Theorem 4 (Noninterference).
If P *is typable in* Γ *then* P *is* Γ*-secure.*

Current and future work

- Investigate alternative semantics for reactive concurrency.
- Extend the result to the distributed reactive language ULM = call-by-value + side-effects + reactiveness + mobility [Bou03].