Information Flow Testing

Gurvan Le Guernic

IRISA - Univerité de Rennes 1
Kansas State University
INRIA-MSR Joint Center
Gurvan.Le_Guernic@irisa.fr

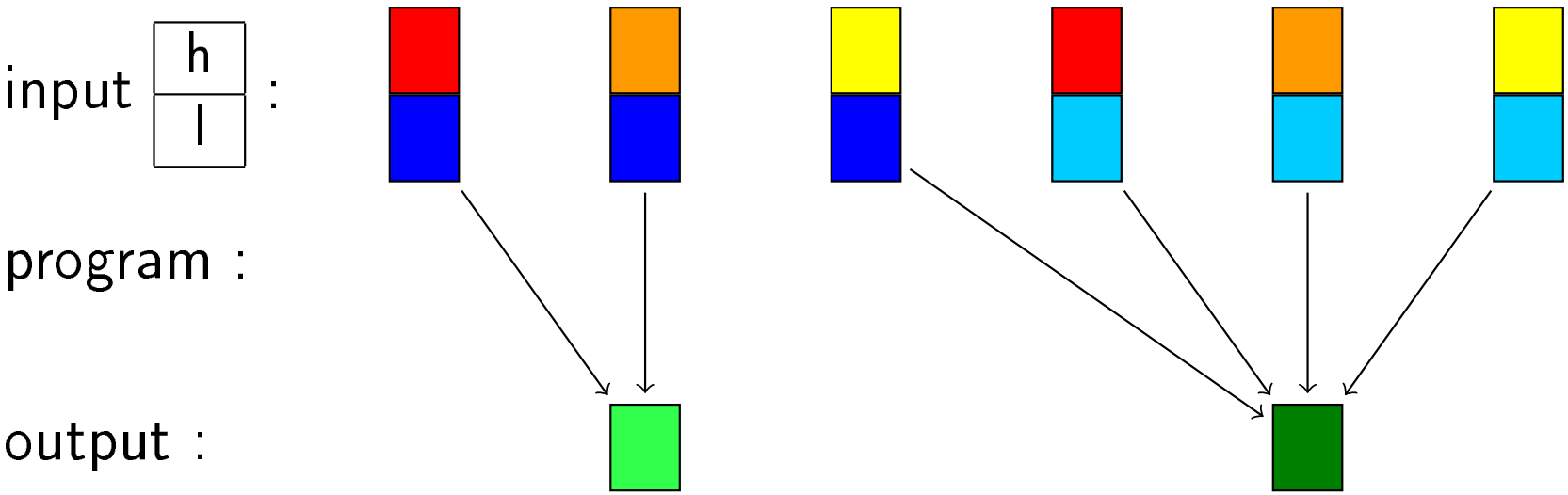
December 9th, 2007 / ASIAN

Outline

Introduction

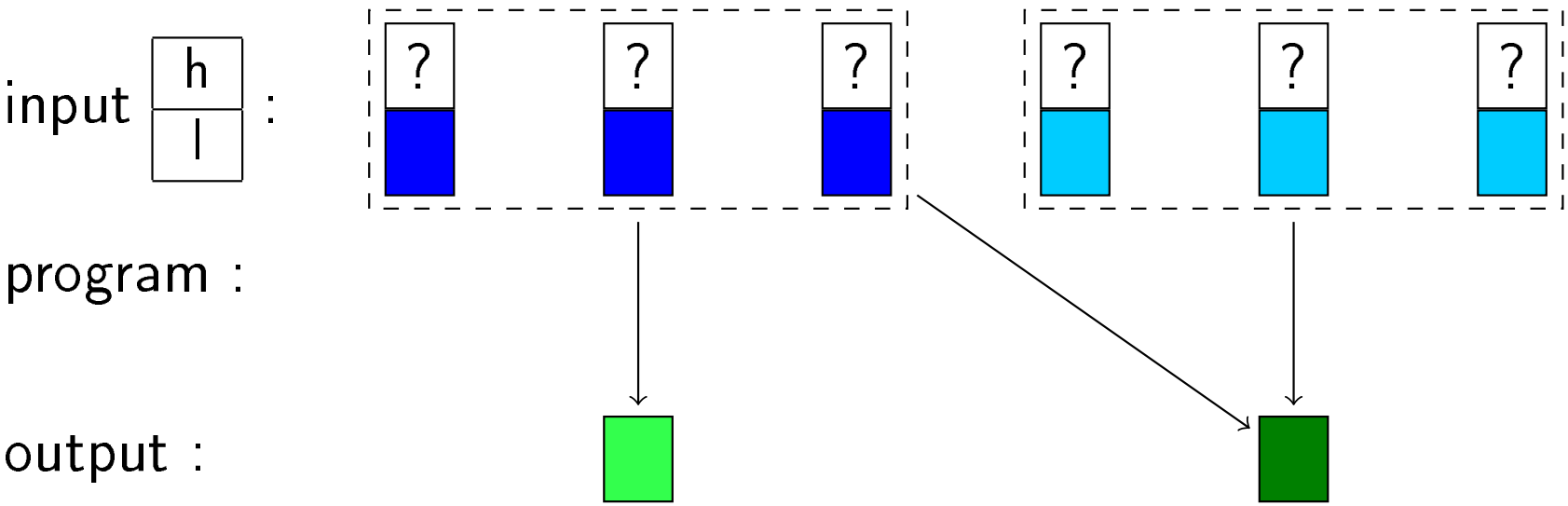
Noninterference

- Cohen (77), Goguen and Meseguer (82)
- Property of a program respecting secrets' confidentiality
- Private (high) inputs do not influence public (low) outputs



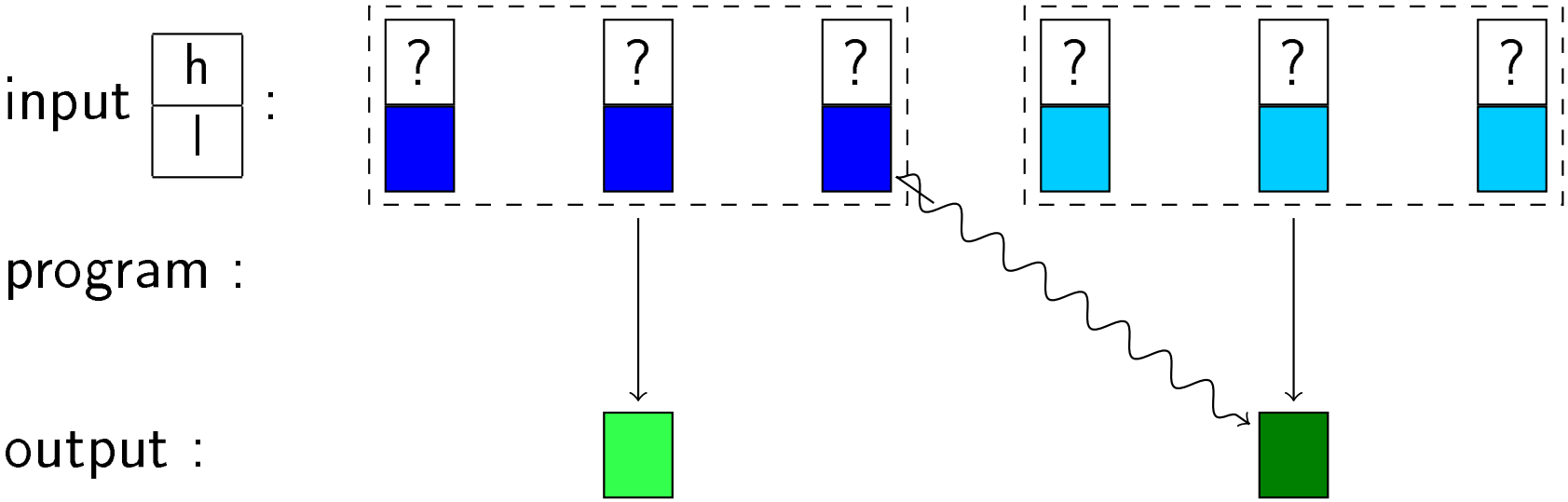
Noninterference

- Cohen (77), Goguen and Meseguer (82)
- Property of a program respecting secrets' confidentiality
- Private (high) inputs do not influence public (low) outputs



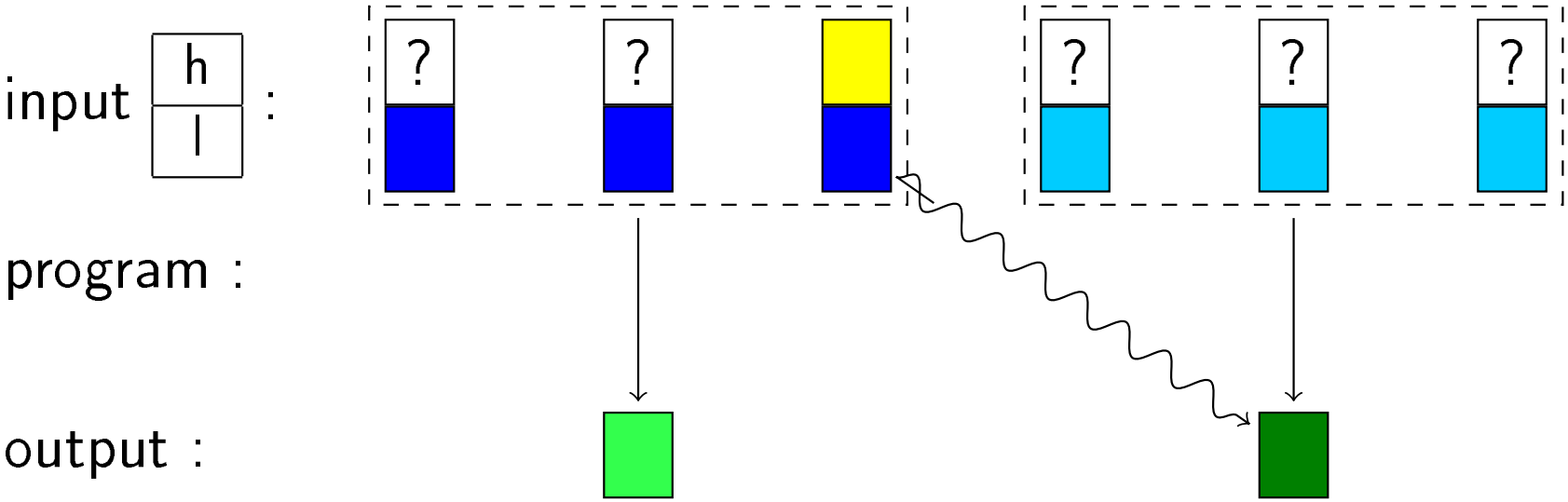
Noninterference

- Cohen (77), Goguen and Meseguer (82)
- Property of a program respecting secrets' confidentiality
- Private (high) inputs do not influence public (low) outputs



Noninterference

- Cohen (77), Goguen and Meseguer (82)
- Property of a program respecting secrets' confidentiality
- Private (high) inputs do not influence public (low) outputs



Noninterfering Execution

Definition 1 (Low Equivalent States: $\zeta_1 \stackrel{V}{=} \zeta_2$)

$\forall$ states ζ_1, resp. ζ_2, containing the value stores σ_1, resp. σ_2:

$$\zeta_1 \stackrel{V}{=} \zeta_2 \iff \forall x \in V : \sigma_1(x) = \sigma_2(x)$$

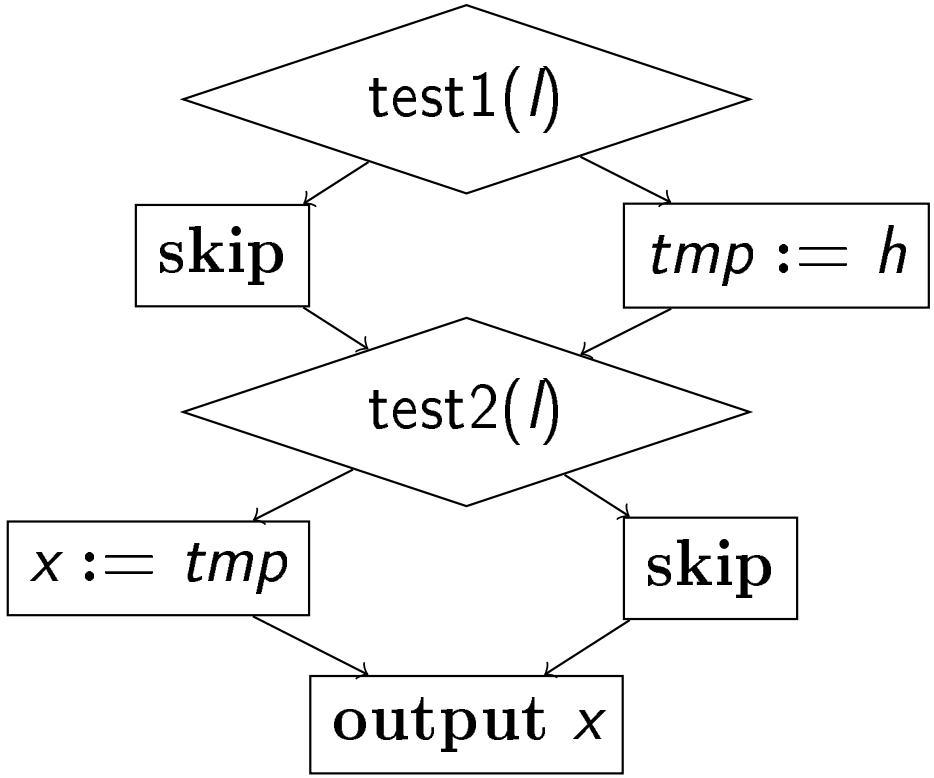
Definition 2 (Noninterfering Execution)

$\forall$ P whose secret input variables are $\mathcal{S}(\mathrm{P})$ and public output variables are $\mathcal{O}(\mathrm{P})$, the execution started in state ζ_1 is noninterfering iff:

$$\forall \zeta_2 : \quad \zeta_1 \stackrel{\overline{\mathcal{S}(\mathrm{P})}}{=} \zeta_2 \quad \Rightarrow \quad [\![\zeta_1 \vdash \mathrm{P}]\!] \stackrel{\mathcal{O}(\mathrm{P})}{=} [\![\zeta_2 \vdash \mathrm{P}]\!]$$

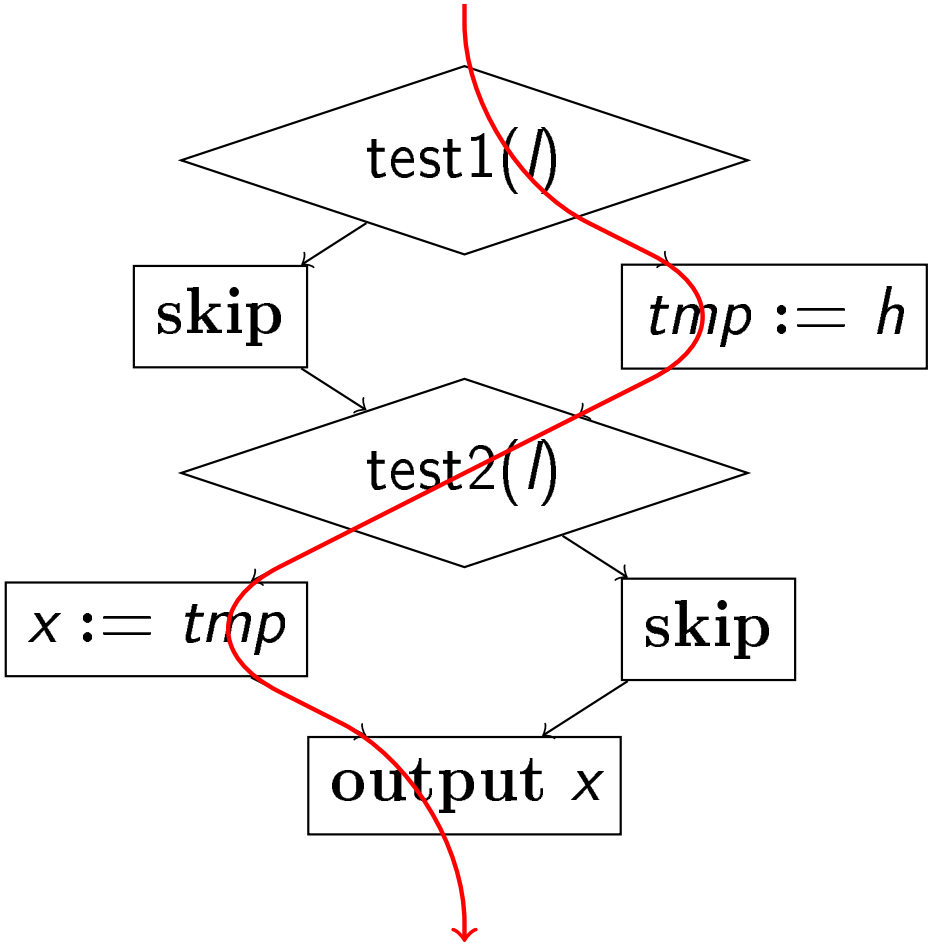
Interest of Dynamic Analyses

h is a private input l is a public input x is a public output



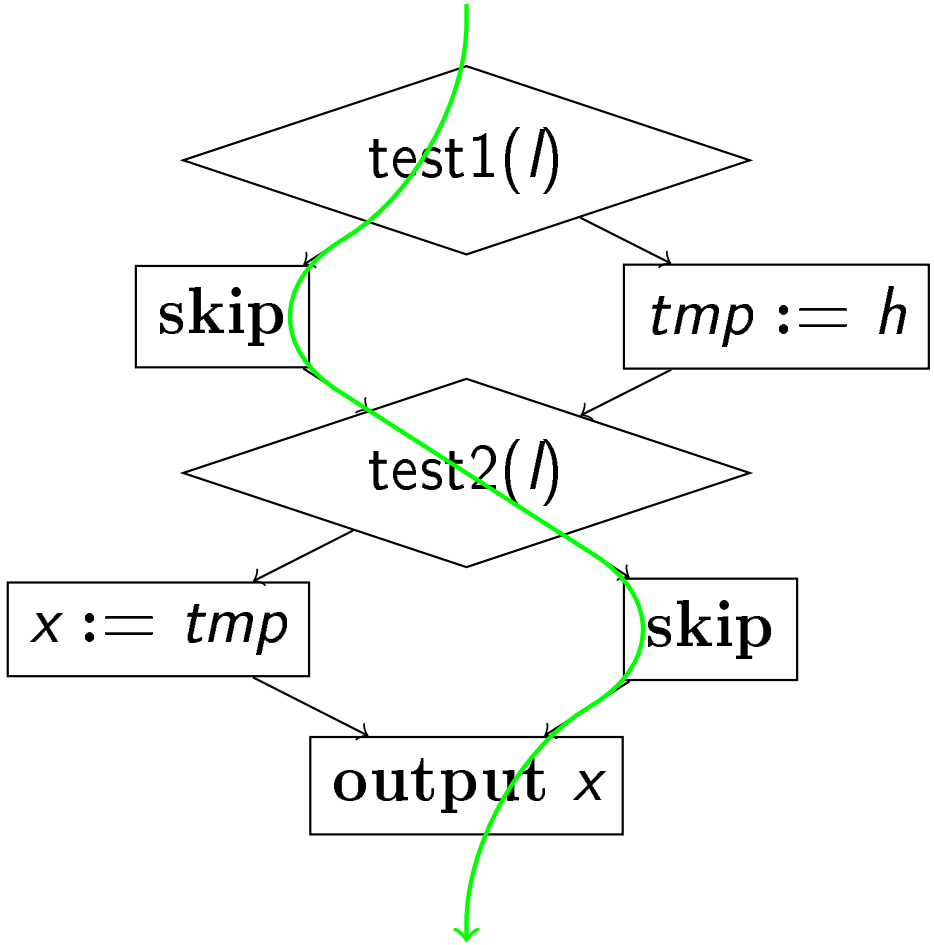
Interest of Dynamic Analyses

h is a private input l is a public input x is a public output



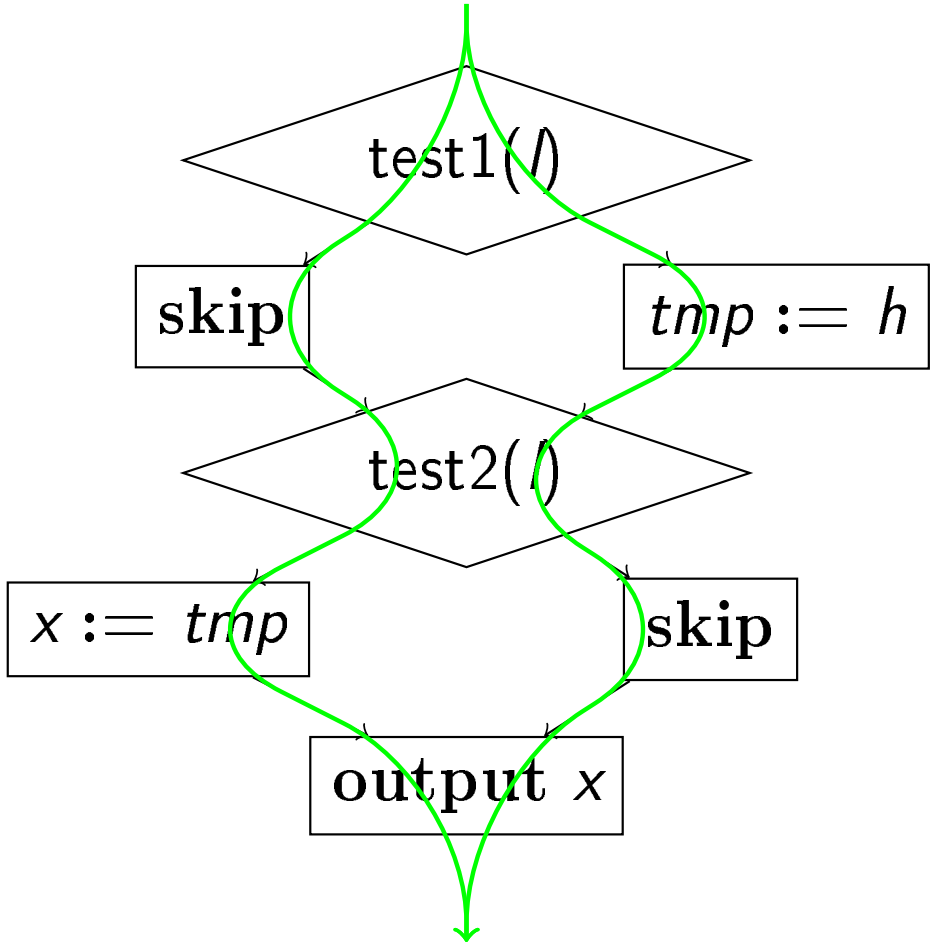
Interest of Dynamic Analyses

h is a private input l is a public input x is a public output



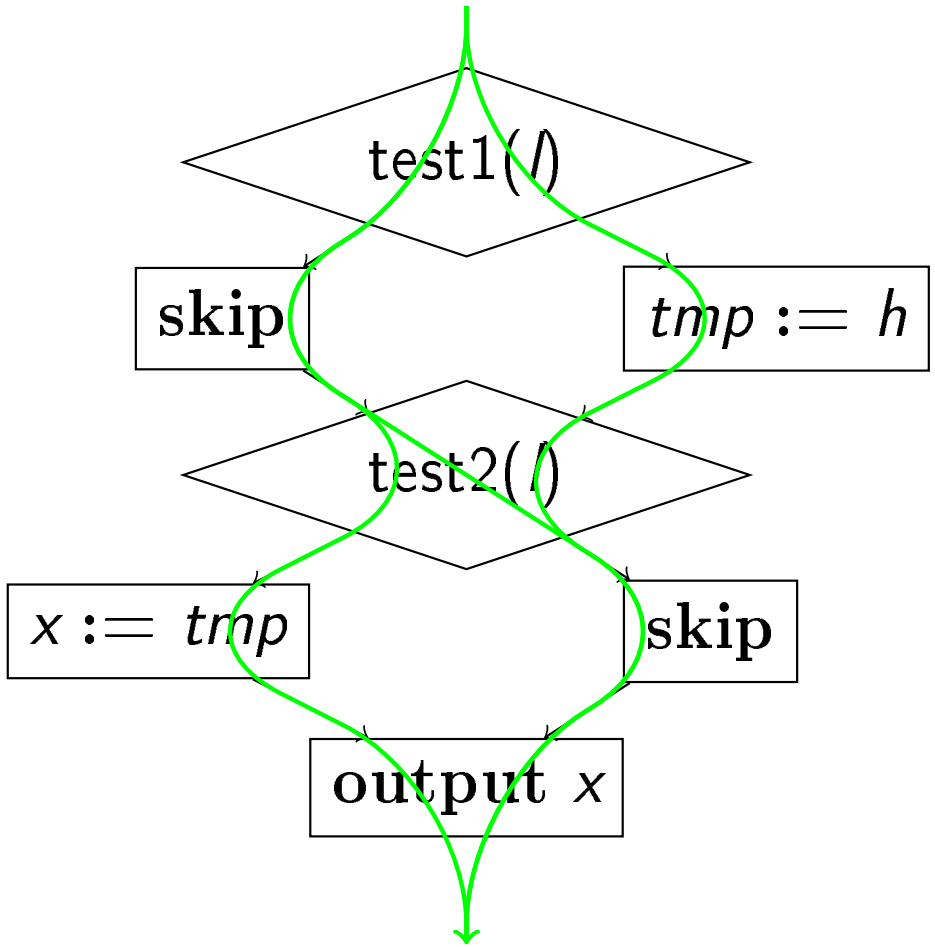
Interest of Dynamic Analyses

h is a private input l is a public input x is a public output



Interest of Dynamic Analyses

h is a private input l is a public input x is a public output



Is Detection Enough for a Monitor?

What happens with an analysis which is *sound* with regard to information flow detection?

- Static analysis:

Expert: "You should not use this program!"

- Run-time analysis:

Is Detection Enough for a Monitor?

What happens with an analysis which is *sound* with regard to information flow detection?

- Static analysis:

Expert: "You should not use this program!"

- Run-time analysis:

ATM: "Oh, by the way, I probably sent your PIN code all over the web."

Is Detection Enough for a Monitor?

What happens with an analysis which is *sound* with regard to information flow detection?

- Static analysis:

Expert: "You should not use this program!"

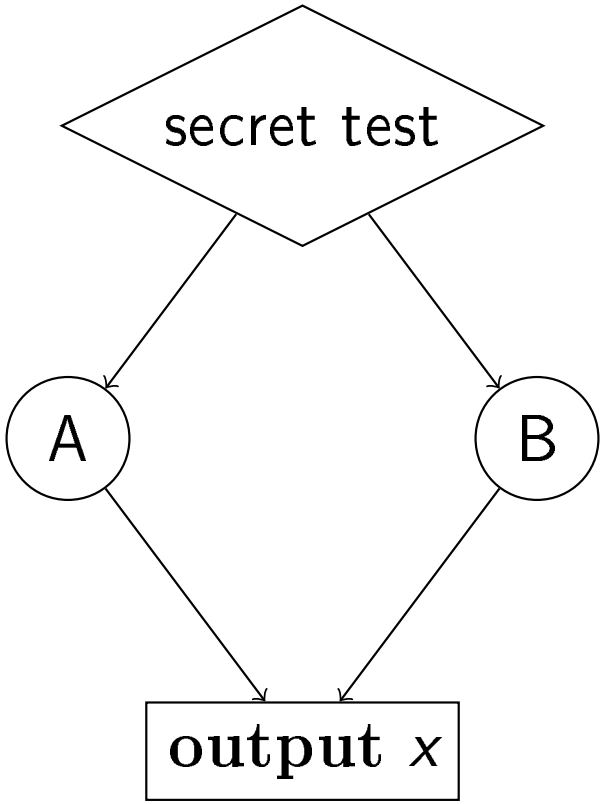
- Run-time analysis:

ATM: "Oh, by the way, I probably sent your PIN code all over the web."

A user expect a noninterference monitor to detect *and correct* information flows.

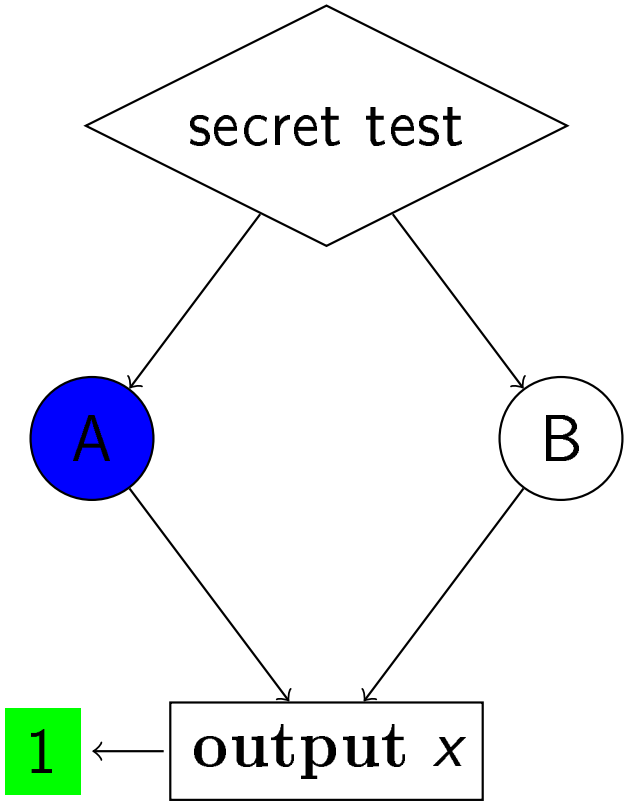
The correction pitfall

public data: ■ secret data: ■



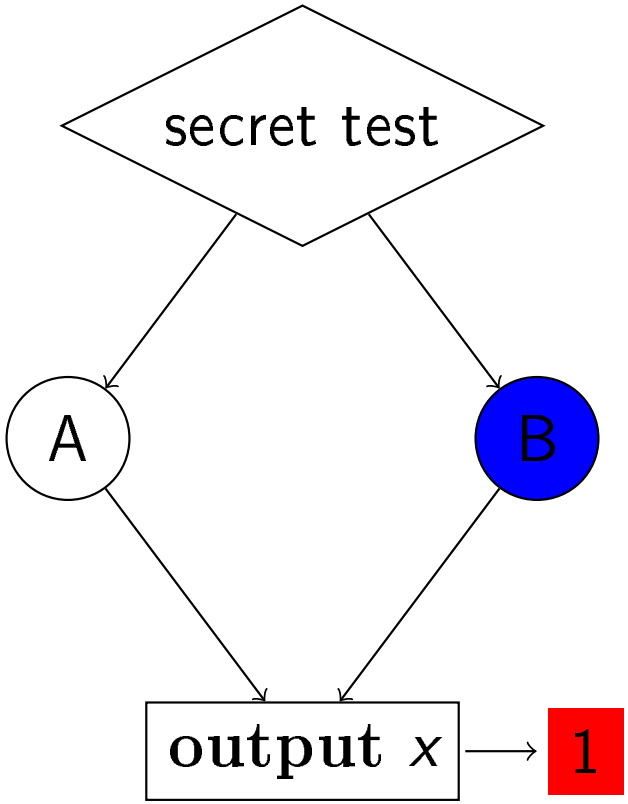
The correction pitfall

public data: ■ secret data: ■



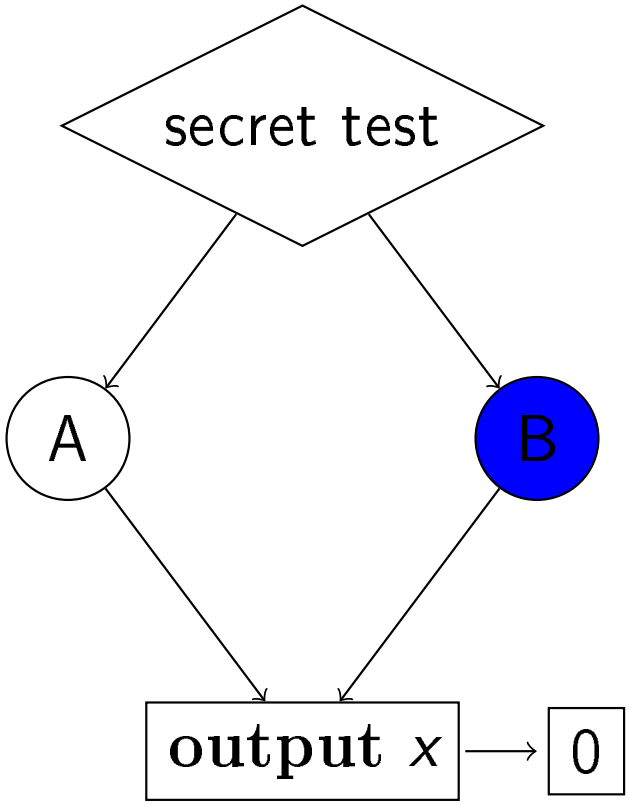
The correction pitfall

public data: ■ secret data: ■



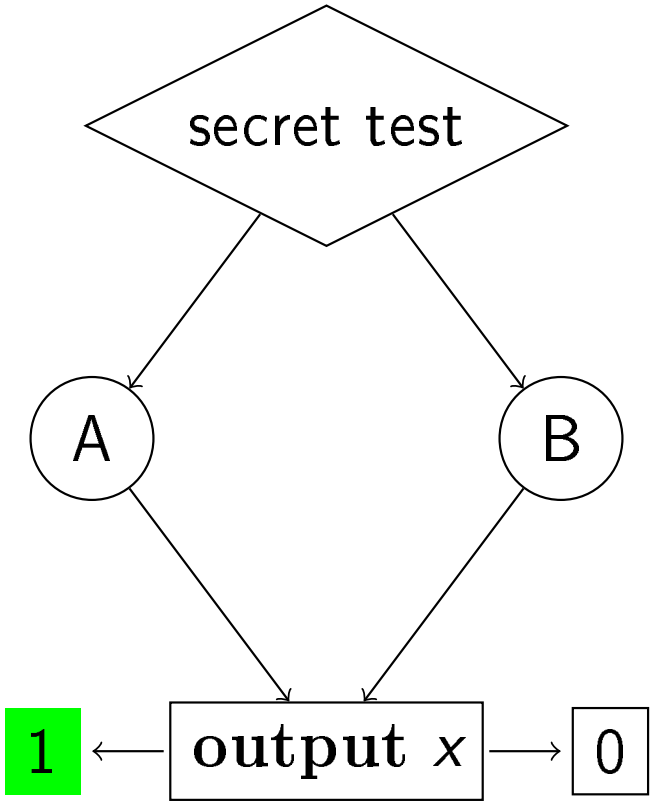
The correction pitfall

public data: ■ secret data: ■



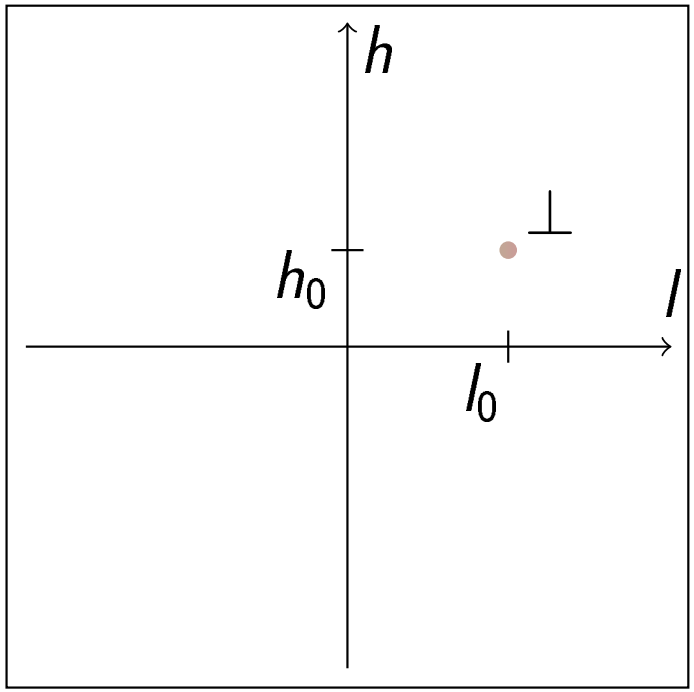
The correction pitfall

public data: ■ secret data: ■



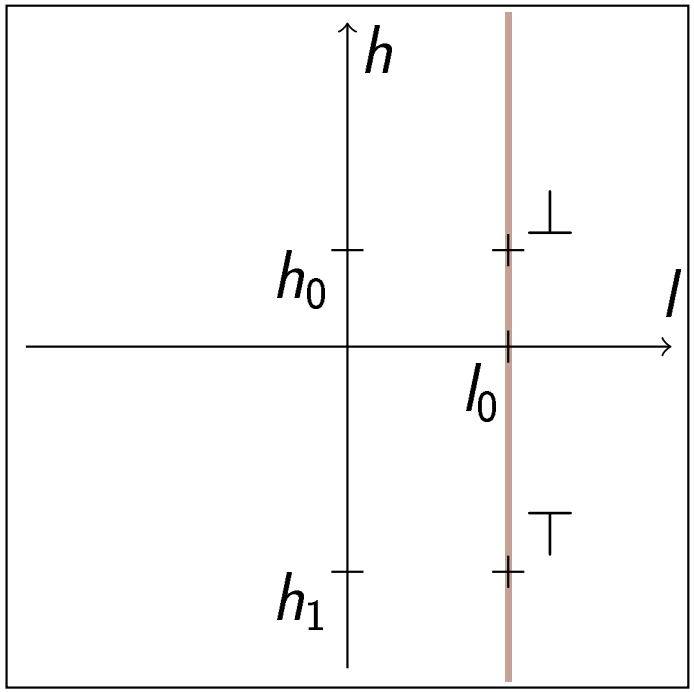
Presentation of the Approach

Main Idea behind Noninterference Testing



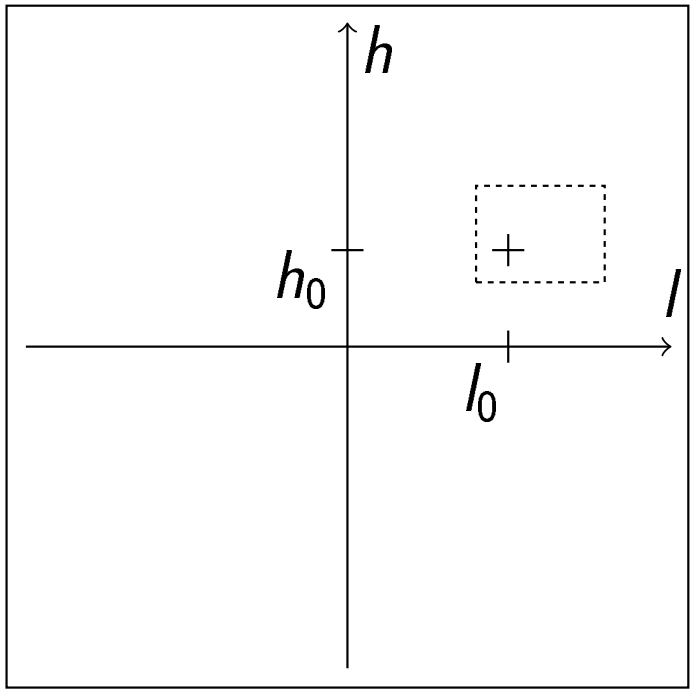
- every point in the plane represents an execution
  - ● : public output of the execution
- coordinates are input values (h: secret inputs, l: public inputs)
- ⊤ and ⊥: noninterference tags
  - ⊤: *may* be influenced by secret inputs
  - ⊥: is *not* influenced by secret inputs

Main Idea behind Noninterference Testing



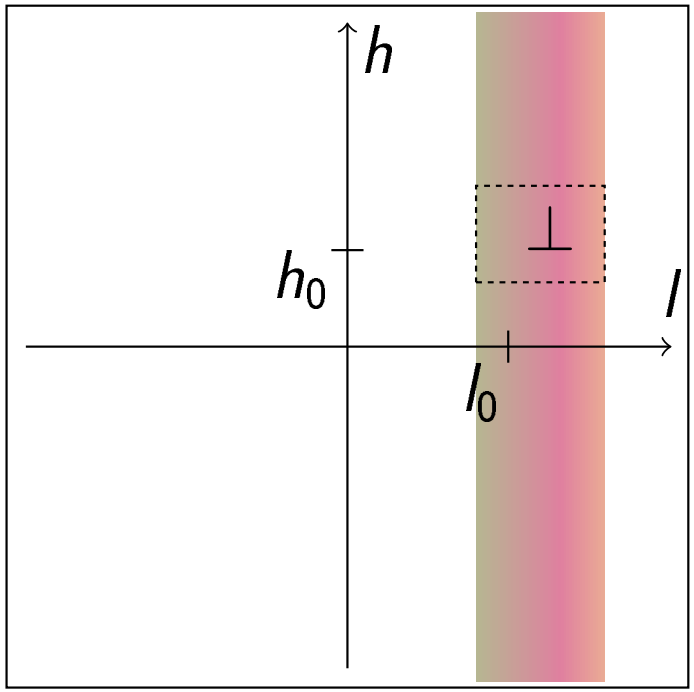
- every point in the plane represents an execution
  - : public output of the execution
- coordinates are input values (h: secret inputs, l: public inputs)
- ⊤ and ⊥: noninterference tags
  - ⊤: *may* be influenced by secret inputs
  - ⊥: is *not* influenced by secret inputs

Main Idea behind Noninterference Testing



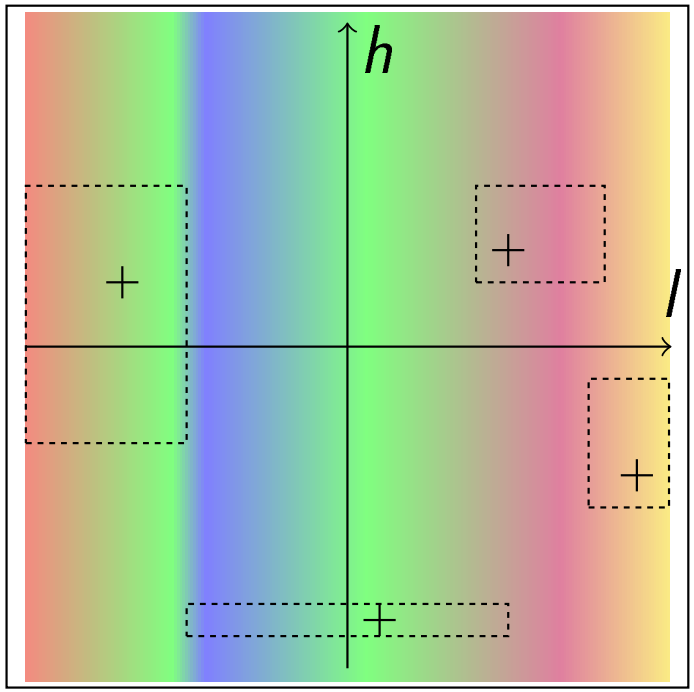
- every point in the plane represents an execution
  - : public output of the execution
- coordinates are input values (h: secret inputs, l: public inputs)
- $\top$ and $\bot$: noninterference tags
  - $\top$: *may* be influenced by secret inputs
  - $\bot$: is *not* influenced by secret inputs

Main Idea behind Noninterference Testing



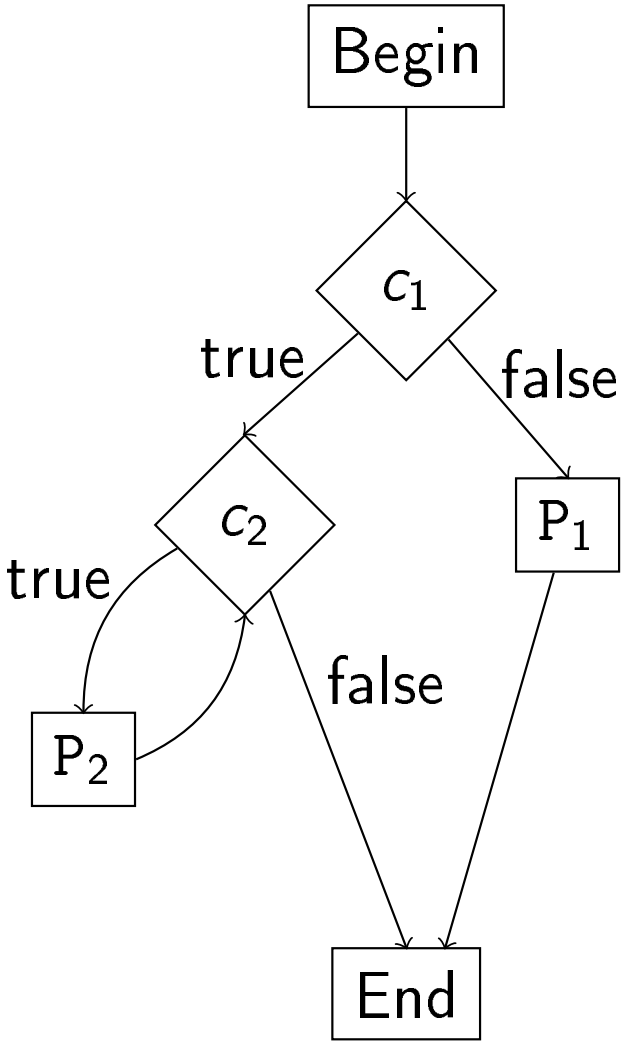
- every point in the plane represents an execution
  - : public output of the execution
- coordinates are input values (h: secret inputs, l: public inputs)
- ⊤ and ⊥: noninterference tags
  - ⊤: *may* be influenced by secret inputs
  - ⊥: is *not* influenced by secret inputs

Main Idea behind Noninterference Testing

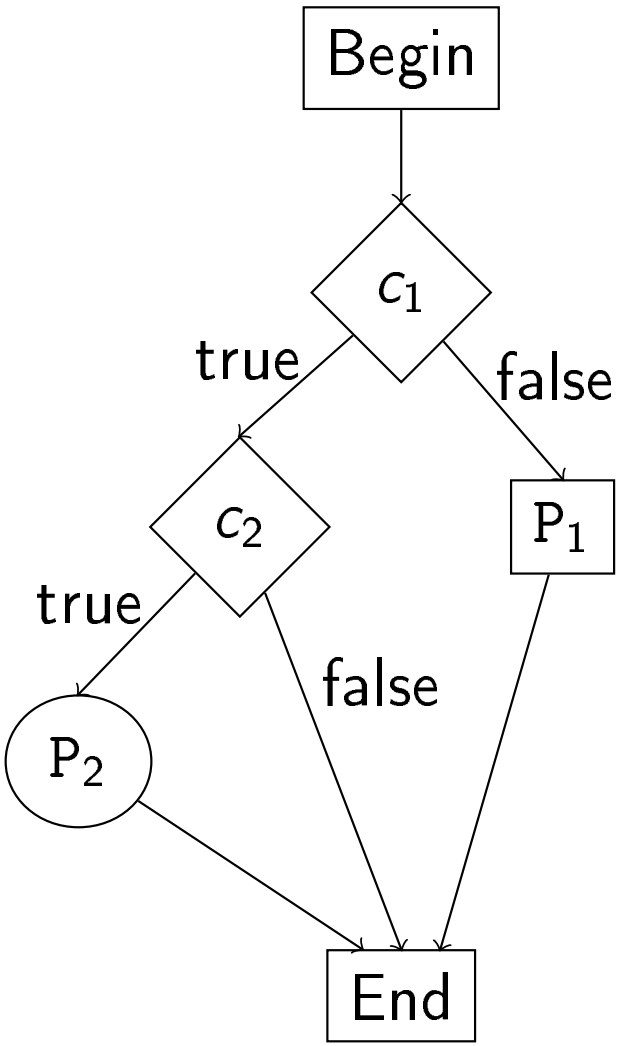


- every point in the plane represents an execution
  - : public output of the execution
- coordinates are input values (h: secret inputs, l: public inputs)
- $\top$ and $\bot$: noninterference tags
  - $\top$: *may* be influenced by secret inputs
  - $\bot$: is *not* influenced by secret inputs

Test Coverage



Test Coverage



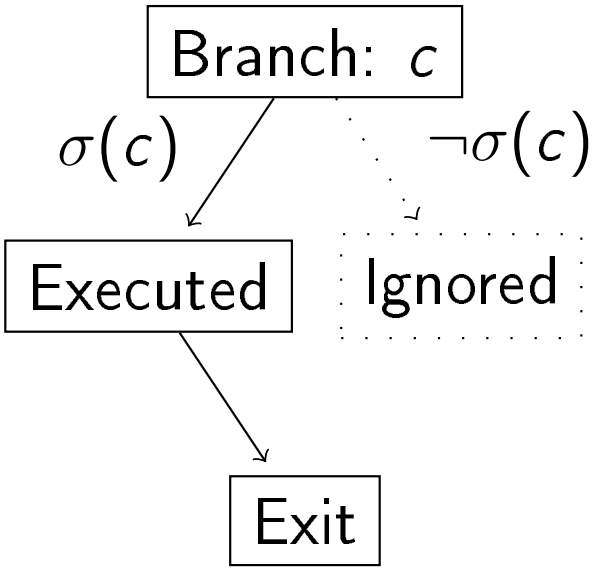
- Noninterference Testing Hypothesis: Every tests following the same path have the same analysis result
- Coverage: every decision combinations "Boundary-interior path coverage": easier to achieve than C2 coverage (every path)

Dynamic Noninterference Analysis

Context

- The language studied is a simple imperative language with loops
- Analysis maintains a tag store identifying variables which *may* be influenced by secret inputs
- Uses a static analysis to take into account *implicit indirect* flows (due to assignments unexecuted)

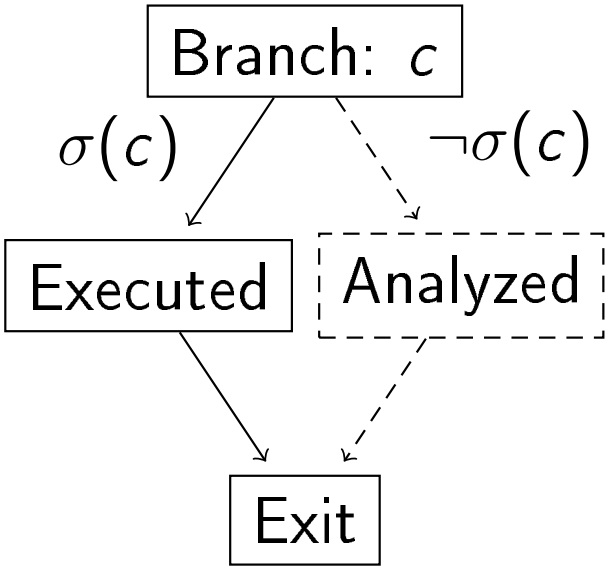
Branching statements



if c is *not* influenced by private inputs

$\Rightarrow$ ignore un-executed branch

Branching statements

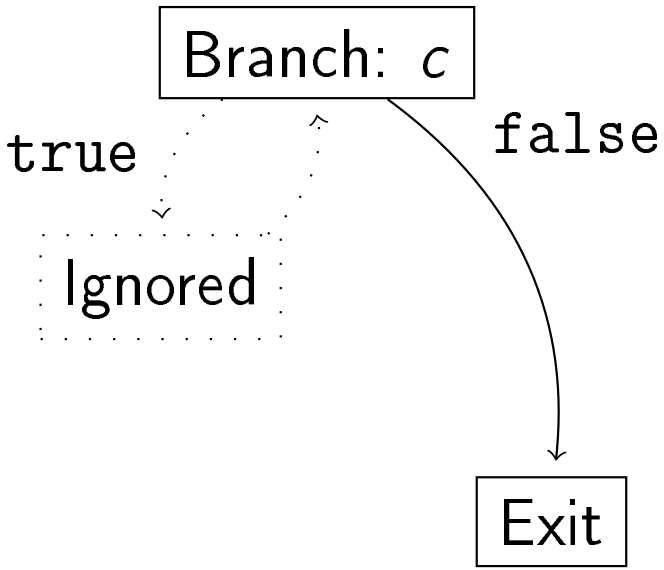


if c is influenced by private inputs

$\Rightarrow$ analyze un-executed branch

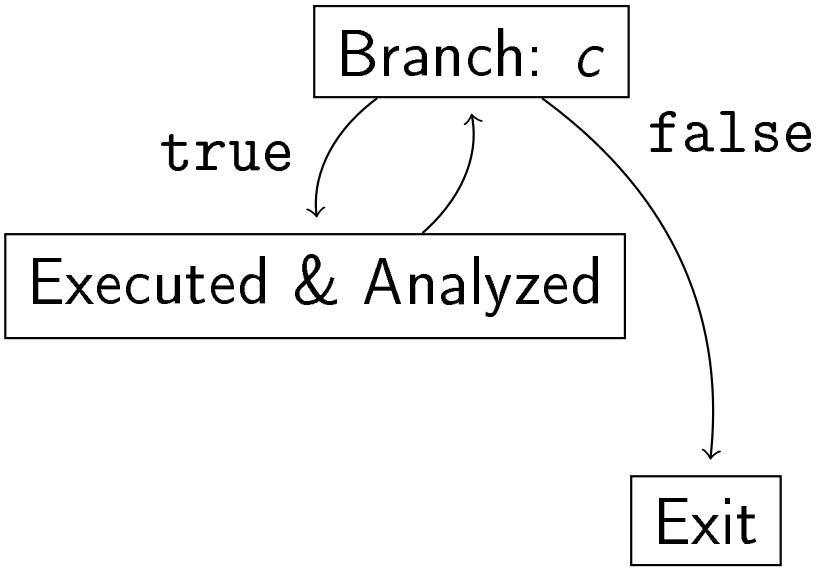
- collect potentially modified variables

Loop statements



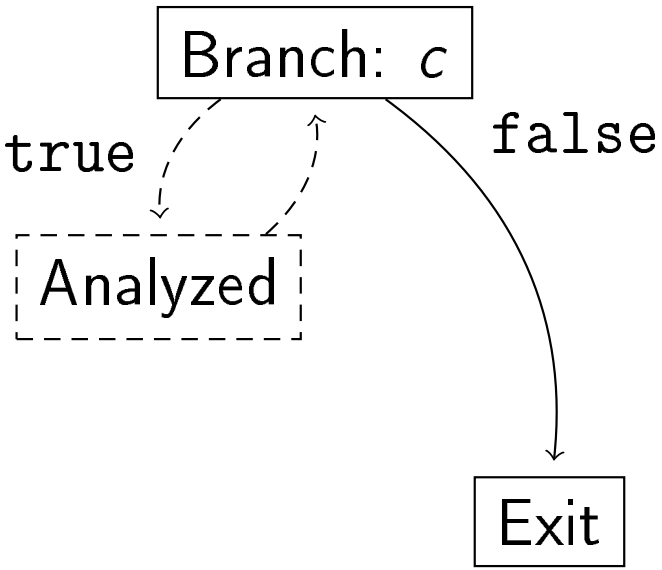
- $\sigma(c)$ is `false`
- c is *not* influenced by private inputs

Loop statements



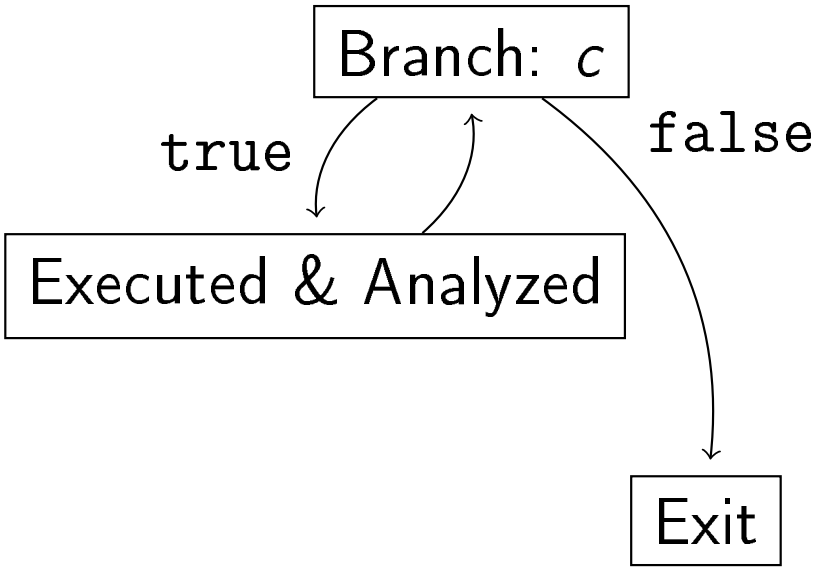
- $\sigma(c)$ is `true`
- c is *not* influenced by private inputs

Loop statements



- $\sigma(c)$ is `false`
- c is influenced by private inputs

Loop statements



- $\sigma(c)$ is `true`
- c is influenced by private inputs

Static Analysis Used

- Returns a set of *potentially assigned* variables and the *dependencies* between initial and final values
- Not precisely defined, instead 3 hypotheses are given
  - Sound detection of modified variables
  - Sound detection of dependencies
  - Deterministic static analysis
- Simple algorithm to extract such analysis from noninterference type systems
- Set of constraints unrelated to the dynamic analysis ensuring the 3 hypotheses
  - an analysis can be extracted from them by fix-point computation

Conclusion

Theorems

Theorem 3 (Soundness)

$$\mathbb{T}[\![\zeta_1 \vdash P]\!](x) = \bot$$
$$\Downarrow$$
$$(\ \forall \zeta_2 : \quad \zeta_1 \overset{\overline{\mathcal{S}(P)}}{=} \zeta_2 \quad \Rightarrow \quad [\![\zeta_1 \vdash P]\!](x) = [\![\zeta_2 \vdash P]\!](x)\ )$$

Theorem 4 (Identical Same Path Analysis Results)

$$\tau[\![\zeta_1 \vdash P]\!] = \tau[\![\zeta_2 \vdash P]\!]$$
$$\Downarrow$$
$$\mathbb{T}[\![\zeta_1 \vdash P]\!] = \mathbb{T}[\![\zeta_2 \vdash P]\!]$$

Conclusion

- If coverage achieved (*finite number of test cases*): conclusion as strong as static analyses
- Interest of noninterference testing:
  - can be more precise than static analyses
  - is not required to be as conservative as noninterference monitors
- Usage:
  - analyze program with a static analysis
  - if it fails, incorporate the static analysis into the dynamic analysis and test the program

Information Flow Testing

Gurvan Le Guernic

IRISA - Univerité de Rennes 1
Kansas State University
INRIA-MSR Joint Center
Gurvan.Le_Guernic@irisa.fr

December 9th, 2007 / ASIAN