



# *Automata-Based Policies for Information Declassification*

Pablo Giambiagi

Swedish Institute of Computer Science

# *The Problem*

- Information Flow properties need to accommodate declassification if they are to be applied in practice.

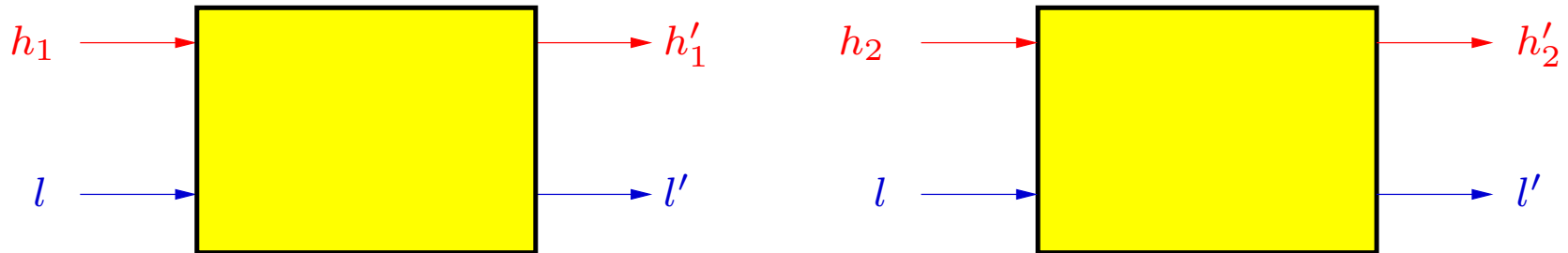
# Approach

- Study declassification of information
- Evolve properties to formalize declassification
- Develop verification methods

# Outline

- Information Flow and Noninterference
- Current Declassification Properties
- Behavioral Declassification Policies
- Baffle Automata
- Wrappers

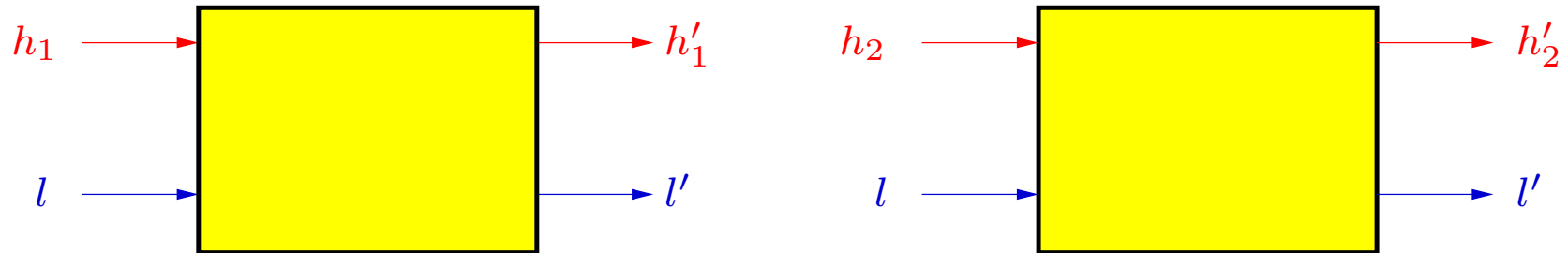
# Information Flow (Noninterference)



Originally:

“Low output is not affected by changes in high input”

# Information Flow (Noninterference)



Originally:

“Low output is not affected by changes in high input”

Better:

“Low behavior is not affected by changes in high behavior”

# *The Evolution of NI*

- Goguen-Meseguer, 1982.
- Basis for most current models of information flow
- Gives semantical foundation for flow type systems
- Initially expressed on ad-hoc deterministic transitions systems
- Later extended to nondeterministic, concurrent systems (with difficulties).
- Probabilistic variants still much in dispute

# Bisimulation-based NI

Bisimulation-based Nondeducibility on Compositions, BNDC  
(Focardi-Gorrieri, 1995).

$$\forall h \in \mathcal{HB} . s \setminus \text{Hi} \approx (s \parallel_{\text{Hi}} h) / \text{Hi}$$

Hi    high-level actions  
 $\mathcal{HB}$     set of high-level processes  
 $\approx$     observational equivalence

# Type systems for NI

- Each piece of data labelled with security class
- Variables typed with security class too.
- Denning (1977), Volpano and Smith (1996), Sabelfeld and Sands (1999, PER model).

Noninterferent program

```
h := ...;  
l := false
```

Interferent program

```
h := ...;  
l := false;  
if h then l := true
```

# Declassification

- Noninterference impedes declassification
- Real applications are expected to release information (declassification).
  - encryption
  - password checking
  - data aggregation (e.g. compute average salary)
  - secret disclosure after certain time

# *Declassification Approaches*



- Relative Secrecy (Volpano and Smith, 2000)
- Admissibility (Giambiagi and Dam, 2000 and 2003)
- Delimited Release (Sabelfeld and Myers, 2004)
- Intransitive noninterference (Mantel and Sands, 2004)
- Abstract noninterference (Giacobazzi and Mastroieni, 2004/5)
- Noninterference “until” (Chong and Myers, 2004)
- Robust Declassification (Myers, Sabelfeld and Zdancewic, 2004)
- Relaxed noninterference (Li and Zdancewic, 2005)
- Declared Information Leakage (Echahed and Prost, 2005)



# *Declassification Policies*

- Implementation independent
- Protocol dependent (?)
- Intensional vs. extensional

# Baffle Automata

A baffle automaton  $\mathcal{B}$  is a *deterministic* transducer

$$\langle B, \Lambda, \Omega, \hookrightarrow, f_0 \rangle$$

- $B$  is the set of automaton states
- $\Lambda$  and  $\Omega$  are respectively the input and output alphabets
- $\hookrightarrow \subseteq B \times (\Lambda \times \Omega) \times B$  is the transition relation
- $f_0 \in B$  is the initial state of the automaton

Input Readiness:  $(\forall f, \alpha)(\exists f', \beta) f \xrightarrow{\alpha/\beta} f'$

Determinism:  $f \xrightarrow{\alpha/\beta_1} f_1 \ \& \ f \xrightarrow{\alpha/\beta_2} f_2 \Rightarrow \beta_1 = \beta_2 \ \& \ f_1 = f_2$

# Baffles for Noninterference

$\mathcal{B}_{NI}$ : Baffle automaton for Noninterference with single state  $f_0$  and transition relation given by

$$\frac{\alpha \in \text{Hi}}{f_0 \xrightarrow{\alpha/\tau} f_0} \quad \frac{\alpha \in \text{Lo} \cup \{\tau\}}{f_0 \xrightarrow{\alpha/\alpha} f_0}$$

Hi: high-level actions

Lo: low-level actions

# Baffle Application

Given the LTS  $\mathcal{S} = \langle S, \Lambda, \rightarrow \rangle$

and the baffle automaton  $\mathcal{B} = \langle B, \Lambda, \Omega, \hookrightarrow, f_0 \rangle$

The *application* of  $\mathcal{B}$  to  $\mathcal{S}$  is the system  $\mathcal{S} \prec \mathcal{B} \succ = \langle S \prec B \succ, \Lambda, \rightarrow_F \rangle$   
with transitions

$$\frac{s \xrightarrow{\alpha} s' \quad f \xrightarrow{\alpha/\beta} f'}{s \prec f \succ \xrightarrow{\beta} s' \prec f' \succ}$$

# Example 1

ProgA: <i>hi</i> := read(); write <i>lo</i>		<i>hi</i> is a high-level variable read() inputs a high-level value <i>lo</i> is a low variable
---	--	---

Two different high-level behaviors:

$$h \triangleq v_0 := \mathbf{read}()$$
$$h' \triangleq v_1 := \mathbf{read}()$$

What can an observer learn from these two systems?

$$ProgA \parallel_{Hi} h$$
$$ProgA \parallel_{Hi} h'$$

# Example I

ProgA:  $\left| \begin{array}{l} h \triangleq v_0 := \mathbf{read}() \\ h' \triangleq v_1 := \mathbf{read}() \end{array} \right.$   
 $hi := \mathbf{read}();$   
 $\mathbf{write} lo$

$ProgA \parallel_{Hi} h \prec \mathcal{B}_{NI} \sim \tau. \mathbf{write} lo \sim ProgA \parallel_{Hi} h' \prec \mathcal{B}_{NI}$

$h \prec \mathcal{B}_{NI} \sim \tau \sim h' \prec \mathcal{B}_{NI}$

ProgA satisfies Noninterference

# Example II

ProgB:  $\left| \begin{array}{l} h \triangleq v_0 := \mathbf{read}() \\ h' \triangleq v_1 := \mathbf{read}() \end{array} \right.$   
 $hi := \mathbf{read}();$   
 $\mathbf{write } hi$

$ProgB \parallel_{Hi} h \prec \mathcal{B}_{NI} \sim \tau. \mathbf{write } v_0$   
 $\not\sim \tau. \mathbf{write } v_1 \sim ProgB \parallel_{Hi} h' \prec \mathcal{B}_{NI}$

ProgB does NOT satisfy Noninterference

# *BNDC with Baffle Automata*

$$\forall h \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx s \setminus \text{Hi}$$

# *BNDC with Baffle Automata*

$$\forall h \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx s \setminus \text{Hi}$$



$$\forall h, h' \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx (s \parallel_{\text{Hi}} h') / \text{Hi}$$

# *BNDC with Baffle Automata*

$$\forall h \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx s \setminus \text{Hi}$$



$$\forall h, h' \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx (s \parallel_{\text{Hi}} h') / \text{Hi}$$



$$\forall h, h' \in \mathcal{HB}. s \parallel_{\text{Hi}} h \langle \mathcal{B}_{NI} \rangle \approx s \parallel_{\text{Hi}} h' \langle \mathcal{B}_{NI} \rangle$$

# *BNDC with Baffle Automata*

$$\forall h \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx s \setminus \text{Hi}$$



$$\forall h, h' \in \mathcal{HB}. (s \parallel_{\text{Hi}} h) / \text{Hi} \approx (s \parallel_{\text{Hi}} h') / \text{Hi}$$



$$\forall h, h' \in \mathcal{HB}. s \parallel_{\text{Hi}} h \prec \mathcal{B}_{NI} \approx s \parallel_{\text{Hi}} h' \prec \mathcal{B}_{NI}$$



$$\forall h, h' \in \mathcal{HB}. h \prec \mathcal{B}_{NI} \approx h' \prec \mathcal{B}_{NI}$$

$$\Rightarrow s \parallel_{\text{Hi}} h \prec \mathcal{B}_{NI} \approx s \parallel_{\text{Hi}} h' \prec \mathcal{B}_{NI}$$

# Baffle Security

$$\forall h, h' \in \mathcal{HB}. \left[ \begin{array}{c} h \langle \mathcal{B} \rangle \sim h' \langle \mathcal{B} \rangle \\ \Downarrow \\ s \parallel_{\text{Hi}} h \langle \mathcal{B} \rangle \sim s \parallel_{\text{Hi}} h' \langle \mathcal{B} \rangle \end{array} \right]$$

# Parity Declassification Example

ProgC:  $hi := \text{read}();$   
 $\text{write}(hi \bmod 2)$  |  $h \triangleq 50 := \text{read}()$   
 $h' \triangleq 41 := \text{read}()$

$\text{ProgC} \parallel_{Hi} h \sim 50 := \text{read}(). \text{write } 0$

$\text{ProgC} \parallel_{Hi} h' \sim 41 := \text{read}(). \text{write } 1$

A Baffle automaton  $\mathcal{B}_1$  for parity declassification:

$$\frac{f \in \mathbb{Z} \cup \{\bullet\}}{f \xrightarrow{v := \text{read}()/*} v} \qquad \frac{n = v \bmod 2}{v \xrightarrow{\text{write } n/\partial} v}$$



# Parity Declassification Ex. II

ProgD:

```
x := read();  
if x < 100 then write (x mod 2)  
else write 1
```

$h \triangleq 50 := \text{read}()$

$h' \triangleq 100 := \text{read}()$

$\text{ProgD} \parallel_{\text{Hi}} h \sim 50 := \text{read}(). \text{write } 0$

$\text{ProgD} \parallel_{\text{Hi}} h' \sim 100 := \text{read}(). \text{write } 1$

$\text{ProgD} \parallel_{\text{Hi}} h \langle \mathcal{B}_1 \rangle \sim *. \partial \not\approx *. \text{write } 1 \sim \text{ProgD} \parallel_{\text{Hi}} h' \langle \mathcal{B}_1 \rangle$

ProgC does NOT satisfy  $\mathcal{B}_1$ -security

# Parity Declassification Ex. III

ProgE:

$x := \text{read}();$

$\text{write } (x+1 \bmod 2)$

$h \triangleq 50 := \text{read}()$

$h' \triangleq 41 := \text{read}()$

$\text{ProgE} \parallel_{\text{Hi}} h \prec \mathcal{B}_1 \succ \sim *. \text{write } 1 \approx *. \text{write } 0 \sim \text{ProgE} \parallel_{\text{Hi}} h' \prec \mathcal{B}_1 \succ$

ProgE does NOT satisfy  $\mathcal{B}_1$ -security  
but leaks as much as the parity

# Parity Declassification Ex. IIIb

ProgE:

$x := \text{read}();$   
 $\text{write } (x+1 \bmod 2)$

$h \triangleq 50 := \text{read}()$

$h' \triangleq 41 := \text{read}()$

Define  $\mathcal{B}_2$  as

$$\frac{v \bmod 2 = 0}{f_0 \xrightarrow{v := \text{read}() / \text{even}} f_0}$$

$$\frac{v \bmod 2 = 1}{f_0 \xrightarrow{v := \text{read}() / \text{odd}} f_0}$$

$h \langle \mathcal{B}_1 \rangle \sim \text{even} \not\sim \text{odd} \sim h' \langle \mathcal{B}_1 \rangle$

ProgE satisfies  $\mathcal{B}_2$ -security

# Crypto Declassification

<b>ProgF:</b>	$h \triangleq 50 := \text{read}()$
$k := \text{pubkey}();$	$h' \triangleq 41 := \text{read}()$
$m := \text{read}();$	
<b>write enc(m, k)</b>	

$\text{ProgF} \parallel_{\text{Hi}} h \sim k := \text{pubkey}(). 50 := \text{read}(). \text{write } \{50\}_k$

$\text{ProgF} \parallel_{\text{Hi}} h' \sim k := \text{pubkey}(). 41 := \text{read}(). \text{write } \{41\}_k$

# Crypto Declassification

<b>ProgF:</b> $k := \text{pubkey}();$ $m := \text{read}();$ $\text{write enc}(m, k)$	$h \triangleq 50 := \text{read}()$ $h' \triangleq 41 := \text{read}()$
---	---

A Baffle automaton  $\mathcal{B}_3$  for cryptographic declassification:

$$\begin{array}{c}
 \frac{m, k \in \mathbb{Z}_\perp \quad m' \in \mathbb{Z}}{(m, k) \xrightarrow{m' := \text{read}()/*} (m', k)} \quad \frac{m, k \in \mathbb{Z}_\perp \quad k' \in \mathbb{Z}}{(m, k) \xrightarrow{k' := \text{pubkey}()/k' := \text{pubkey}()} (m, k')} \\
 \\
 \frac{m, k \in \mathbb{Z} \quad n = \{m\}_k}{(m, k) \xrightarrow{\text{write } n/\partial} (m, k)}
 \end{array}$$

# Crypto Declassification

ProgF:

$k := \text{pubkey}();$

$m := \text{read}();$

$\text{write enc}(m, k)$

$h \triangleq 50 := \text{read}()$

$h' \triangleq 41 := \text{read}()$

$\text{ProgF} \parallel_{\text{Hi}} h \prec \mathcal{B}_3 \succ \sim k := \text{pubkey}(). * .\partial \sim \text{ProgF} \parallel_{\text{Hi}} h' \prec \mathcal{B}_3 \succ$

ProgF satisfies  $\mathcal{B}_3$ -security

# Crypto Declassification II

ProgG:

$k := \text{pubkey}();$

$m := \text{read}();$

$\text{write enc}(m, 317)$

$h \stackrel{\Delta}{=} 50 := \text{read}()$

$h' \stackrel{\Delta}{=} 41 := \text{read}()$

$\text{ProgG} \parallel_{\text{Hi}} h \prec \mathcal{B}_3 \succ \sim k := \text{pubkey}(). * .\text{write } \{50\}_{317}$

$\text{ProgG} \parallel_{\text{Hi}} h' \prec \mathcal{B}_3 \succ \sim k := \text{pubkey}(). * .\text{write } \{41\}_{317}$

ProgG does NOT satisfy  $\mathcal{B}_3$ -security

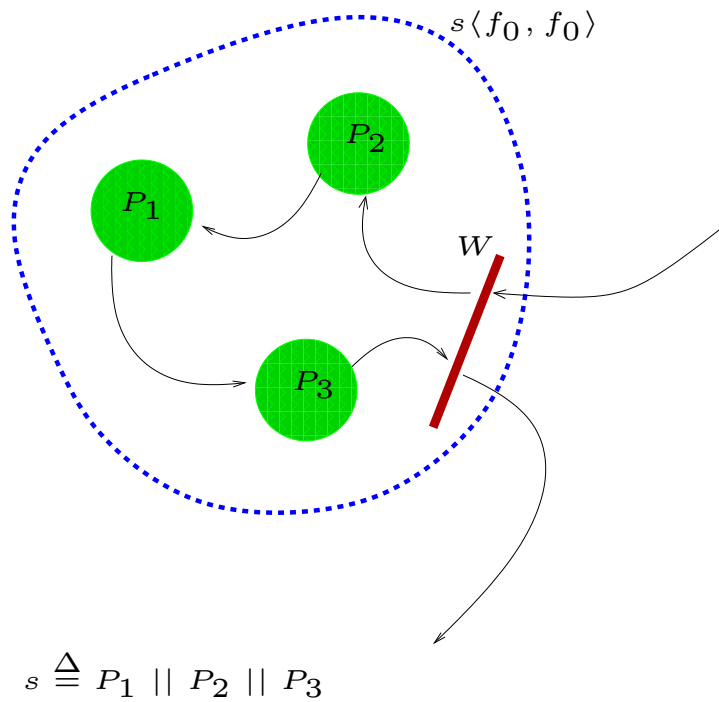
# Verification Difficulties

Baffle security:

$$\forall h, h' \in \mathcal{HB}. \left[ \begin{array}{c} h \langle \mathcal{B} \rangle \sim h' \langle \mathcal{B} \rangle \\ \Downarrow \\ s \parallel_{\text{Hi}} h \langle \mathcal{B} \rangle \sim s \parallel_{\text{Hi}} h' \langle \mathcal{B} \rangle \end{array} \right]$$

Quantifies universally over high-level behaviors.

# Wrappers



Some transitions of the baffle automaton

$$f_0 \xrightarrow{a/x} f_1 \xrightarrow{b/y} f_2$$

$$f_0 \xrightarrow{a'/x} f_3 \xrightarrow{b'/y} f_4$$

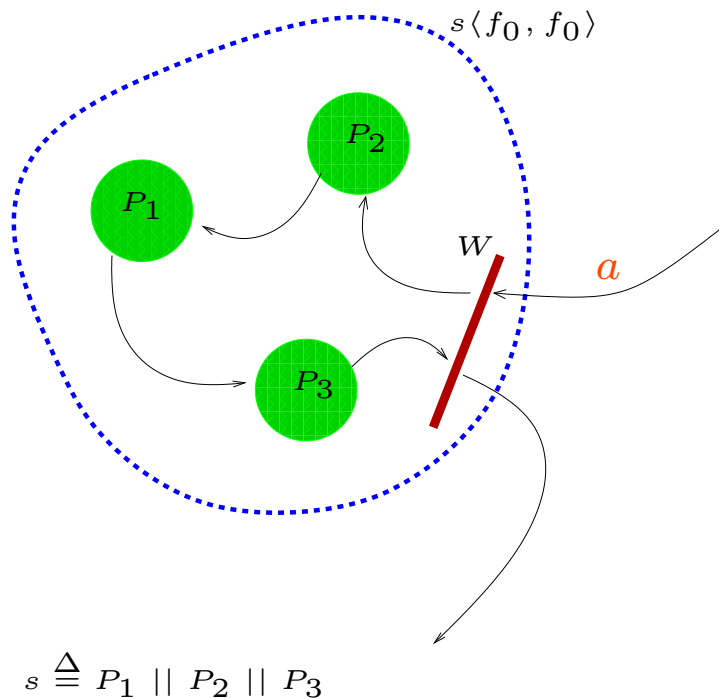
Internal behavior:

$s$

External behavior:

$$s \langle f_0, f_0 \rangle$$

# Wrappers



Some transitions of the baffle automaton

$$f_0 \xrightarrow{a/x} f_1 \xrightarrow{b/y} f_2$$

$$f_0 \xrightarrow{a'/x} f_3 \xrightarrow{b'/y} f_4$$

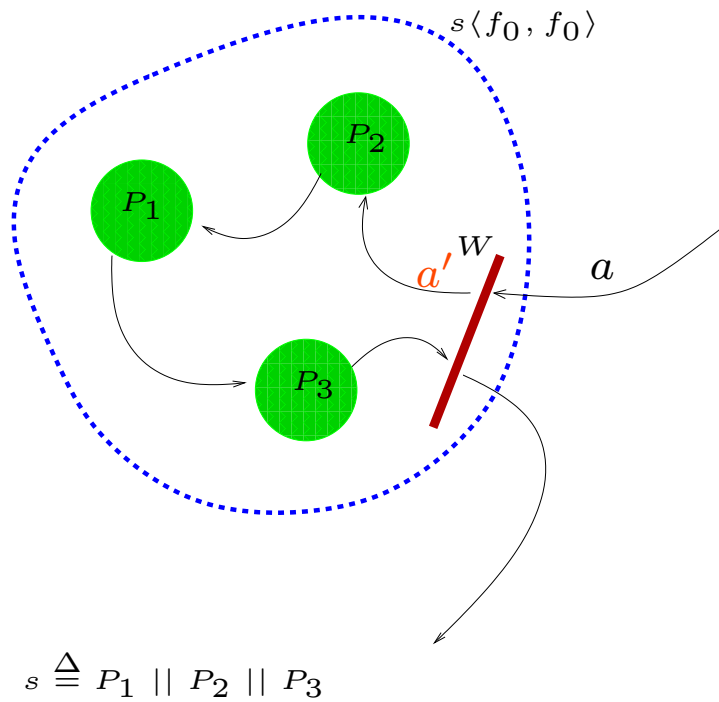
Internal behavior:

$s$

External behavior:

$$s \langle f_0, f_0 \rangle \xrightarrow{a}$$

# Wrappers



Some transitions of the baffle automaton

$$f_0 \xrightarrow{a/x} f_1 \xrightarrow{b/y} f_2$$

$$f_0 \xrightarrow{a'/x} f_3 \xrightarrow{b'/y} f_4$$

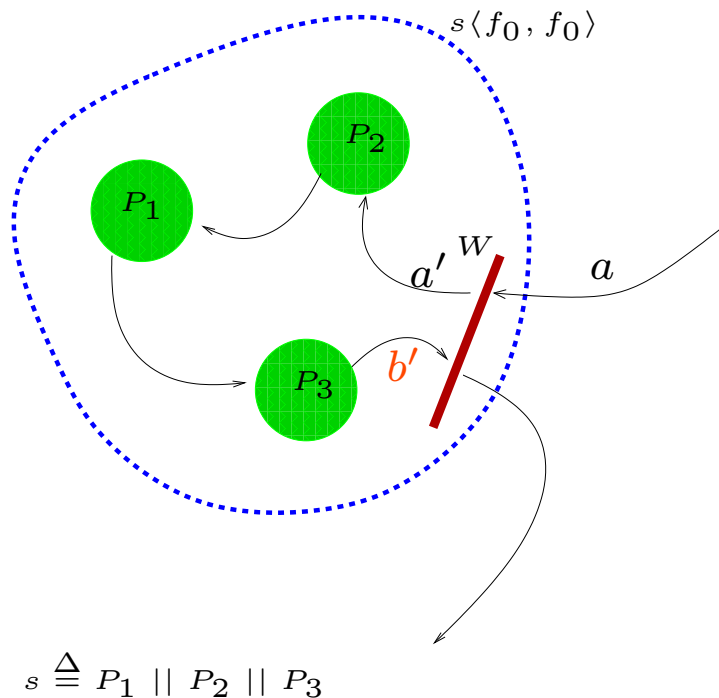
Internal behavior:

$$s \xrightarrow{a'} s_1$$

External behavior:

$$s \langle f_0, f_0 \rangle \xrightarrow{a} s_1 \langle f_3, f_1 \rangle$$

# Wrappers



Some transitions of the baffle automaton

$$f_0 \xrightarrow{a/x} f_1 \xrightarrow{b/y} f_2$$

$$f_0 \xrightarrow{a'/x} f_3 \xrightarrow{b'/y} f_4$$

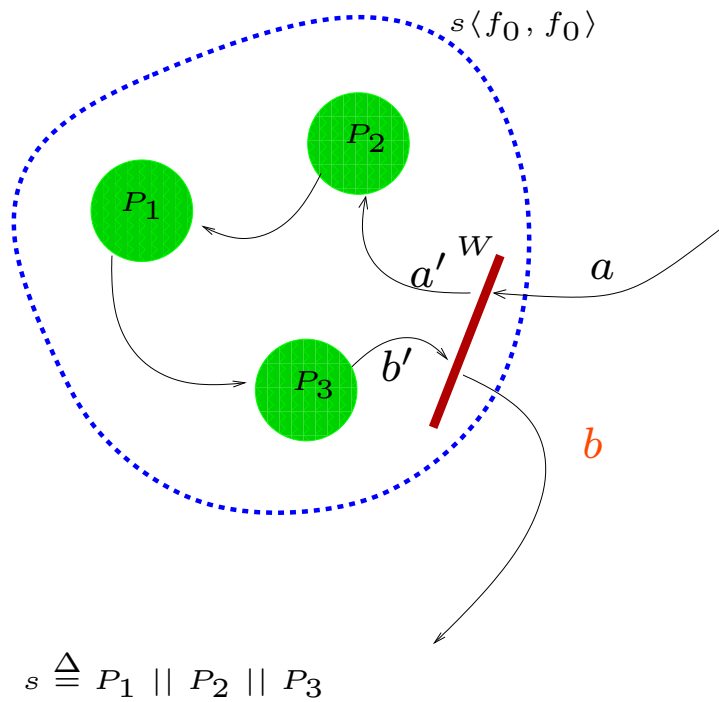
Internal behavior:

$$s \xrightarrow{a'} s_1 \xrightarrow{b'} s_2$$

External behavior:

$$s \langle f_0, f_0 \rangle \xrightarrow{a} s_1 \langle f_3, f_1 \rangle$$

# Wrappers



Some transitions of the baffle automaton

$$f_0 \xrightarrow{a/x} f_1 \xrightarrow{b/y} f_2$$

$$f_0 \xrightarrow{a'/x} f_3 \xrightarrow{b'/y} f_4$$

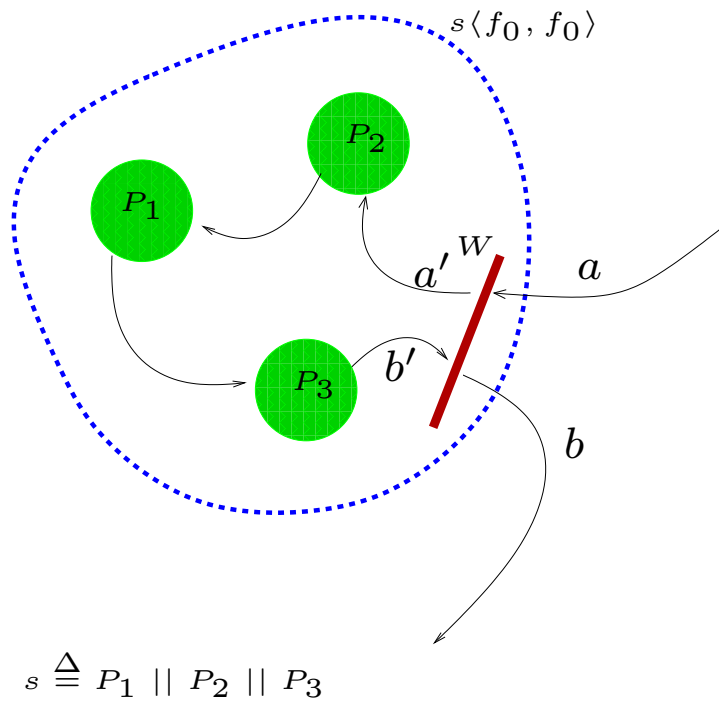
Internal behavior:

$$s \xrightarrow{a'} s_1 \xrightarrow{b'} s_2$$

External behavior:

$$s \langle f_0, f_0 \rangle \xrightarrow{a} s_1 \langle f_3, f_1 \rangle \xrightarrow{b} s_2 \langle f_4, f_2 \rangle$$

# Wrappers



Transitions of the wrapped system:

$$\frac{s \xrightarrow{\alpha'} s' \quad f_1 \xrightarrow{\alpha'/\beta} f'_1 \quad f_2 \xrightarrow{\alpha/\beta} f'_2}{s \langle f_1, f_2 \rangle \xrightarrow{\alpha} s' \langle f'_1, f'_2 \rangle}$$



# *Wrappers Imply Baffle-Security*

$f_0$ : initial state of  $\mathcal{B}$ .

Main result: **If  $s\langle f_0, f_0 \rangle \sim s$ , then  $s$  is  $\mathcal{B}$ -secure**

# Wrappers Imply Baffle-Security

$f_0$ : initial state of  $\mathcal{B}$ .

Main result: **If  $s \langle f_0, f_0 \rangle \sim s$ , then  $s$  is  $\mathcal{B}$ -secure**

... provided:

- $\mathcal{B}$  is low-insensitive (no state changes on low-actions)
- $\mathcal{B}$  is *output uniform* ( $t$  deadlocks iff  $t \langle f_1, f_2 \rangle$  deadlocks)

# Wrappers Imply Baffle-Security

$f_0$ : initial state of  $\mathcal{B}$ .

Main result: **If  $s \langle f_0, f_0 \rangle \sim s$ , then  $s$  is  $\mathcal{B}$ -secure**

... provided:

- $\mathcal{B}$  is low-insensitive (no state changes on low-actions)
- $\mathcal{B}$  is *output uniform* ( $t$  deadlocks iff  $t \langle f_1, f_2 \rangle$  deadlocks)

but it avoids the universal quantification over high-level behaviors

# Using Wrappers

- Verify  $s \sim s\langle f_0, f_0 \rangle$  directly?

# Using Wrappers

- Verify  $s \sim s\langle f_0, f_0 \rangle$  directly?
- Type system to enforce  $s \sim s\langle f_0, f_0 \rangle$ ?

# Using Wrappers

- Verify  $s \sim s\langle f_0, f_0 \rangle$  directly?
- Type system to enforce  $s \sim s\langle f_0, f_0 \rangle$ ?
- Runtime enforcement:

If  $\mathcal{B}$  is output-uniform, then  $s\langle f_0, f_0 \rangle \sim (s\langle f_0, f_0 \rangle)\langle f_0, f_0 \rangle$

# Using Wrappers

- Verify  $s \sim s\langle f_0, f_0 \rangle$  directly?
- Type system to enforce  $s \sim s\langle f_0, f_0 \rangle$ ?
- Runtime enforcement:

If  $\mathcal{B}$  is output-uniform, then  $s\langle f_0, f_0 \rangle \sim (s\langle f_0, f_0 \rangle)\langle f_0, f_0 \rangle$

**Corollary:**

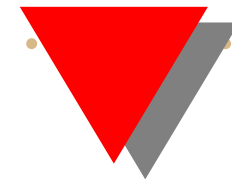
The wrapped system  $s\langle f_0, f_0 \rangle$  satisfies  $\mathcal{B}$ -security

# Creol

- An object-oriented experimental programming language
- Distributed concurrent objects
- Asynchronous method calls
- Processor release points
- Executable operational semantics written in *Maude*

<http://www.ifi.uio.no/~creol/>

# Creol - Configurations

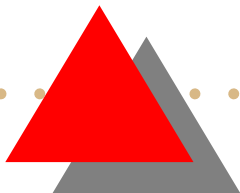


```
sort Configuration .
```

```
subsorts Object MMsg Queue Class < Configuration .
```

```
op none : -> Configuration [ctor] .
```

```
op __ : Configuration Configuration -> Configuration  
      [ctor assoc comm id: none] .
```



# Creol - Object Creation

```
C < o' : Ob | Cl: C' # V, Pr: x:=new(C);r,  
      PrQ: W, Lvar: L, Att: A >
```

=>

```
C (new(C))  
< o' : Ob | Cl: C' # V, Pr: r, PrQ: W, Lvar: L, Att: A >
```

=>

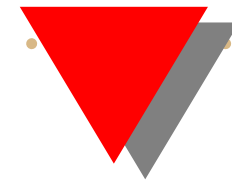
```
C  
< o : Ob | Cl: C, Pr: run, PrQ: none, Lvar: no, Att: no >  
findAttr(_,_,_)  
< o' : Ob | Cl: C' # V, Pr: r, PrQ: W, Lvar: L, Att: A >
```

# *SafeNew – Work in progress*

## Goals:

- add a construct to create wrapped (sets of) objects
- express and compose baffle automata

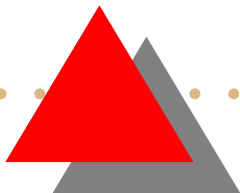
# SafeNew - Wrappers



```
mod WRAPPER is
  pr FLOW-AUTOMATA .

  sorts SecConf Wrapper .
  subsorts Wrapper < Configuration .
  subsorts Object MMsg Queue FAut < SecConf .

  op none : -> SecConf [ctor] .
  op ___ : SecConf SecConf -> SecConf [ctor assoc
                                         comm id: none] .
  op { _ } : SecConf -> Wrapper .
```



# SafeNew - Object Creation

```
C < o' : Ob | Cl: C' # V, Pr: x:=safeNew(C,F);r,  
      PrQ: W, Lvar: L, Att: A >
```

=>

```
C (safeNew(C, F))  
< o' : Ob | Cl: C' # V, Pr: r, PrQ: W, Lvar: L, Att: A >
```

=>

```
C  
{ < o : Ob | Cl: C, Pr: run, PrQ: none, Lvar: no, Att: no >  
  F  
}  
findAttr(_,_,_)  
< o' : Ob | Cl: C' # V, Pr: r, PrQ: W, Lvar: L, Att: A >
```

# *SafeNew - Challenges*

- Attacker model: what is observable? Messages? Class definitions?
- Object creation within or outside the wrapper?
- Method code transportation (from class def'n to processor)

Tack!