

# State-oriented noninterference for CCS

ILARIA CASTELLANI

INRIA SOPHIA ANTIPOLIS

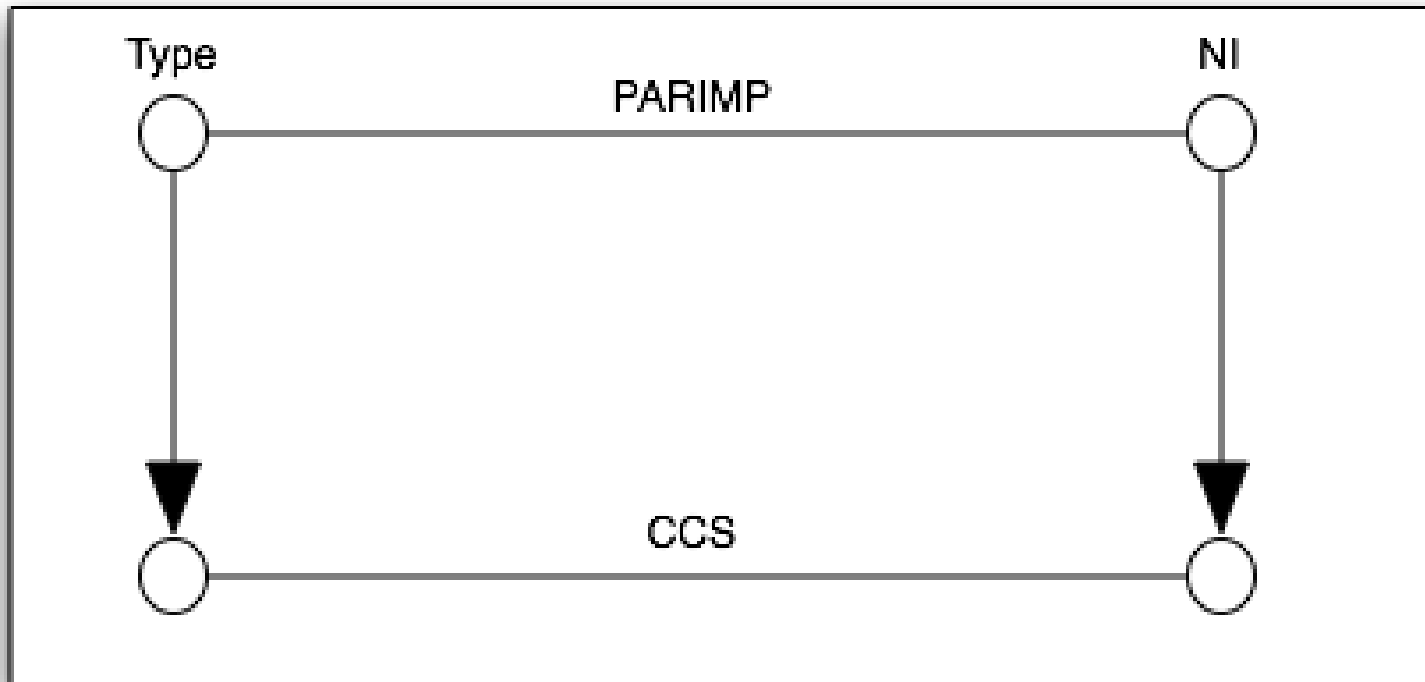
2ème réunion PARSEC

Paris, 20 juin 2007

# Motivation

- Relate language-based security and process calculi security.
- First objective: relate the noninterference property (NI) for a **parallel imperative language** with security properties for **CCS**.
- Starting point:
  - [Focardi, Rossi & Sabelfeld '05]: translation of a **sequential imperative language** into CCS, preserving *time-sensitive* NI.
  - [Honda, Yoshida, Vasconcelos '01] and following papers: translation of more powerful languages into a variant of the  **$\pi$ -calculus**, preserving both NI and types.

# Objective



Translate a parallel imperative language PARIMP into the process calculus CCS, preserving both **noninterference** (NI) and **types**.

# Language-based security

- Information: contained in “objects”, used by “subjects”.
- Objects have **security levels**, eg: high = secret, low = public.
- **Secure information flow**: no flow from high to low objects.

$X_L := Y_H$  not secure

$Z_H := Y_H ; X_L := 0$  secure

- **Imperative languages**:
  - Subjects = programs. Objects = variables. Tools:
  - **(self-)bisimulation** to formalise the security property;
  - **type systems** to statically ensure it.

## Process calculi security

- Subjects = processes. Objects = channels  $a, b, c \dots$

$a_h(x). \bar{b}_\ell \langle x \rangle$  not secure

- **Data flow** and **control flow** are closely intertwined:

$a_h(x). \bar{b}_\ell$                        $a_h(x). \bar{b}_\ell \langle v \rangle$                       secure?

Warning ! Can be used to implement **indirect insecure flows**:

$( a_h(x). \text{if } x \text{ then } \bar{b}_\ell \text{ else } \bar{c}_\ell \mid ( b_\ell. \bar{d}_\ell \langle 0 \rangle + c_\ell. \bar{d}_\ell \langle 1 \rangle ) ) \setminus \{b_\ell, c_\ell\}$

# The imperative language PARIMP

Variables  $X, Y, Z$ , values  $V, V'$  and expressions  $E, E'$ :

$$E ::= F(X_1, \dots, X_n)$$

**Syntax** of programs (or commands)  $C, D$ :

$$C, D ::= \text{nil} \mid X := E \mid C ; D \mid (\text{if } E \text{ then } C \text{ else } D) \mid \\ (\text{while } E \text{ do } C) \mid (C \parallel D)$$

**Semantics**: transitions on configurations  $\langle C, s \rangle \rightarrow \langle C', s' \rangle$  where  $s, s'$  are *states* (finite mappings from variables to values).

## Operational semantics of PARIMP (1/3)

$$\text{(ASSIGN-OP)} \quad \frac{}{\langle X := E, s \rangle \rightarrow \langle \text{nil}, s[s(E)/X] \rangle}$$

$$\text{(SEQ-OP1)} \quad \frac{\langle C, s \rangle \rightarrow \langle C', s' \rangle}{\langle C; D, s \rangle \rightarrow \langle C'; D, s' \rangle}$$

$$\text{(SEQ-OP2)} \quad \frac{}{\langle \text{nil}; D, s \rangle \rightarrow \langle D, s \rangle}$$

## Operational semantics of PARIMP (2/3)

$$\text{(COND-OP1)} \quad \frac{s(E) = tt}{\langle \text{if } E \text{ then } C \text{ else } D, s \rangle \rightarrow \langle C, s \rangle}$$

$$\text{(COND-OP2)} \quad \frac{s(E) \neq tt}{\langle \text{if } E \text{ then } C \text{ else } D, s \rangle \rightarrow \langle D, s \rangle}$$

$$\text{(WHILE-OP1)} \quad \frac{s(E) = tt}{\langle \text{while } E \text{ do } C, s \rangle \rightarrow \langle C; \text{while } E \text{ do } C, s \rangle}$$

$$\text{(WHILE-OP2)} \quad \frac{s(E) \neq tt}{\langle \text{while } E \text{ do } C, s \rangle \rightarrow \langle \text{nil}, s \rangle}$$

## Operational semantics of PARIMP (3/3)

$$\text{(PARL-OP1)} \quad \frac{\langle C, s \rangle \rightarrow \langle C', s' \rangle}{\langle C \parallel D, s \rangle \rightarrow \langle C' \parallel D, s' \rangle}$$

$$\text{(PARL-OP2)} \quad \frac{}{\langle \text{nil} \parallel D, s \rangle \rightarrow \langle D, s \rangle}$$

$$\text{(PARR-OP1)} \quad \frac{\langle D, s \rangle \rightarrow \langle D', s' \rangle}{\langle C \parallel D, s \rangle \rightarrow \langle C \parallel D', s' \rangle}$$

$$\text{(PARR-OP2)} \quad \frac{}{\langle C \parallel \text{nil}, s \rangle \rightarrow \langle C, s \rangle}$$

## Security property for PARIMP

Variables: partitioned into  $L$  (low variables) and  $H$  (high variables).

$L$ -equality on states:

$s =_L t$  if  $\text{dom}(s) = \text{dom}(t)$  and  $(X \in \text{dom}(s) \cap L \Rightarrow s(X) = t(X))$

$L$ -bisimulation on programs:

Symmetric relation  $\mathcal{S} \subseteq (\mathcal{C} \times \mathcal{C})$  such that  $C \mathcal{S} D$  implies, for any  $s$  and  $t$  such that  $s =_L t$ :

if  $\langle C, s \rangle \rightarrow \langle C', s' \rangle$ , then there exist  $D', t'$  such that

$\langle D, t \rangle \mapsto \langle D', t' \rangle$  where  $s' =_L t'$  and  $C' \mathcal{S} D'$

where  $\mapsto$  is the reflexive closure of  $\rightarrow$  (at most one step).

## Security property for PARIMP (ctd)

*L*-bisimilarity:  $C \simeq_L D$  if  $C \mathcal{S} D$  for some *L*-bisimulation  $\mathcal{S}$ .

*L*-security: a program  $C$  is *L*-secure if  $C \simeq_L C$ .

Examples of insecure programs:

1. `(while  $x_H$  do nil);  $y_L := 0$`

2. `if  $x_H = 0$  then loop ( $y_L := 0$ ;  $y_L := 1$ )  
else loop ( $y_L := 1$ ;  $y_L := 0$ )`

where `loop  $C \stackrel{\text{def}}{=} (\text{while } tt \text{ do } C)$` .

## The process calculus CCS (core)

Process prefixes:  $\pi ::= a(x) \mid \bar{a}\langle e \rangle \mid a \mid \bar{a}$

Parametric processes:  $T ::= A \mid (\mathbf{rec} A(\tilde{x}).P)$

Syntax of CCS processes:

$$P, Q ::= \sum_{i \in I} \pi_i.P_i \mid (P \mid Q) \mid (\nu a)P \mid T(\tilde{e})$$

Abbreviations:

$$\mathbf{0} \stackrel{\text{def}}{=} \sum_{i \in \emptyset} \pi_i.P_i \quad \pi_1.P_1 + \pi_2.P_2 \stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} \pi_i.P_i$$

## Semantics of CCS (1/3)

Actions  $\alpha, \beta, \gamma$ :

$$Act \stackrel{\text{df}}{=} \{av : a \in \mathcal{N}, v \in Val\} \cup \{\bar{a}v : a \in \mathcal{N}, v \in Val\} \cup \{\tau\}$$

Operational rules for **nondeterministic choice**:

$$\text{(SUM-OP}_1\text{)} \quad \sum_{i \in I} \pi_i.P_i \xrightarrow{av} P_i\{v/x\}, \quad \text{if } \pi_i = a(x) \text{ and } v \in Val$$

$$\text{(SUM-OP}_2\text{)} \quad \sum_{i \in I} \pi_i.P_i \xrightarrow{\bar{a}v} P_i, \quad \text{if } \pi_i = \bar{a}\langle e \rangle \text{ and } val(e) = v$$

## Semantics of CCS (2/3)

Operational rules for **parallelism**, **restriction** and **recursion**:

$$\text{(PAR-OP}_1\text{)} \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{(PAR-OP}_2\text{)} \quad \frac{P \xrightarrow{\alpha} P'}{Q \mid P \xrightarrow{\alpha} Q \mid P'}$$

$$\text{(PAR-OP}_3\text{)} \quad \frac{P \xrightarrow{av} P' \quad Q \xrightarrow{\bar{a}v} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{(RES-OP)} \quad \frac{P \xrightarrow{\alpha} P' \quad b \neq \text{subj}(\alpha)}{(\nu b)P \xrightarrow{\alpha} (\nu b)P'}$$

$$\text{(REC-OP)} \quad \frac{P\{\tilde{v}/\tilde{x}\}\{(\text{rec } A(\tilde{x}) . P) / A\} \xrightarrow{\alpha} P' \quad \tilde{v} = \text{val}(\tilde{e})}{(\text{rec } A(\tilde{x}) . P)(\tilde{e}) \xrightarrow{\alpha} P'}$$

## Security properties for CCS

Weak transitions:

- $P \xrightarrow{\alpha} P' \stackrel{\text{df}}{=} P \xrightarrow{\tau}^* \xrightarrow{\alpha} \xrightarrow{\tau}^*$
- $P \xrightarrow{\hat{\alpha}} P' \stackrel{\text{df}}{=} \begin{cases} P \xrightarrow{\alpha} P' & \text{if } \alpha \neq \tau \\ P \xrightarrow{\tau}^* P' & \text{if } \alpha = \tau \end{cases}$

Weak bisimulation:

Symmetric relation  $\mathcal{S} \subseteq (\mathcal{Pr} \times \mathcal{Pr})$  such that  $P \mathcal{S} Q$  implies:

if  $P \xrightarrow{\alpha} P'$  then there exists  $Q'$  such that  $Q \xrightarrow{\hat{\alpha}} Q'$  and  $P' \mathcal{S} Q'$ .

Weak bisimilarity:  $P \approx Q$  if  $P \mathcal{S} Q$  for some weak bisimulation  $\mathcal{S}$ .

## Simple security (BNDC) [Focardi-Gorrieri '95]

Channels are partitioned into **high channels**  $\mathcal{H}$  and low channels  $\mathcal{L}$ .

$\mathcal{Pr}_{\text{syn}}^{\mathcal{H}}$ : set of syntactically high processes, with no channels in  $\mathcal{L}$ .

### Bisimulation-based Non Deducibility on Compositions (BNDC)

$P$  is **secure** with respect to  $\mathcal{H}$ ,  $P \in \text{BNDC}_{\mathcal{H}}$ , if for every  $\Pi \in \mathcal{Pr}_{\text{syn}}^{\mathcal{H}}$ :

$$(\nu\mathcal{H})(P \mid \Pi) \approx (\nu\mathcal{H})P$$

Examples.

$a_h . \bar{b}_\ell$	$a_h + \bar{b}_\ell$	not secure
$a_h \mid \bar{b}_\ell$	$a_h . \bar{b}_\ell + \bar{b}_\ell$	secure

Choosing  $\Pi = \bar{a}_h$  for the first two, we get  $(\nu\mathcal{H})(P \mid \Pi) \not\approx (\nu\mathcal{H})P$ .

## A more robust security property

Transitions  $\xrightarrow{\tilde{\alpha}}_{\mathcal{H}}$ , allowing simulation of high actions by inaction:

$$P \xrightarrow{\tilde{\alpha}}_{\mathcal{H}} P' \stackrel{\text{df}}{=} \begin{cases} P \xrightarrow{\hat{\alpha}} P' \text{ or } P \xrightarrow{\tau}^* P' & \text{if } \text{subj}(\alpha) \in \mathcal{H} \\ P \xrightarrow{\hat{\alpha}} P' & \text{otherwise} \end{cases}$$

**Weak bisimulation up-to-high:**

Symmetric relation  $\mathcal{S} \subseteq (\mathcal{Pr} \times \mathcal{Pr})$  such that  $P \mathcal{S} Q$  implies:

if  $P \xrightarrow{\alpha} P'$  then there exists  $Q'$  such that  $Q \xrightarrow{\tilde{\alpha}}_{\mathcal{H}} Q'$  and  $P' \mathcal{S} Q'$ .

**Weak bisimilarity up to high:**  $P \approx_{\mathcal{H}} Q$  if  $P \mathcal{S} Q$  for some weak bisimulation up to high  $\mathcal{S}$ .

## Persistent security (PBNDC) [Focardi-Rossi '02]

### Persistent BNDC (PBNDC)

$P$  is persistently secure wrt  $\mathcal{H}$ ,  $P \in \text{PBNDC}_{\mathcal{H}}$ , if  $P \approx_{\mathcal{H}} (\nu\mathcal{H})P$ .

Theorem [Focardi-Rossi '02].

$P \in \text{PBNDC}_{\mathcal{H}}$  iff  $P' \in \text{BNDC}$  for any reachable state  $P'$  of  $P$ .

Example.

$P = P_1 + P_2 = a_{\ell}.b_h.\bar{c}_{\ell} + a_{\ell}.(\nu d_{\ell})(d_{\ell} \mid \bar{d}_{\ell} \mid d_{\ell}.\bar{c}_{\ell})$  is secure but not persistently secure.

Secure: show that  $(\nu\mathcal{H})(P \mid \bar{b}_h) \approx (\nu\mathcal{H})P$ .

Not persistently secure: the reachable state  $b_h.\bar{c}_{\ell}$  is not secure.

## A security type system for PBNDC

Inspired from Pottier's type system for the  $\pi$ -calculus (Pottier '02).

**Security levels**  $\sigma, \delta, \theta$  form a **lattice**  $(\mathcal{T}, \leq)$ , where  $\leq$  stands for “less secret than”. Here we assume  $\mathcal{T} = \{\ell, h\}$ , with  $\ell \leq h$ .

**Type environment**  $\Gamma$ : mapping from channels to security levels.

**Type judgements**:  $\Gamma \vdash_{\sigma} P$ .

Intuition:  $\sigma$  is a **lower bound** on the security level of **channels** in  $P$ .

## Typing rules

(SUM)

$$\frac{\forall i \in I : \Gamma(\pi_i) = \sigma \quad \Gamma \vdash_{\sigma} P_i}{\Gamma \vdash_{\sigma} \sum_{i \in I} \pi_i.P_i}$$

(PAR)

$$\frac{\Gamma \vdash_{\sigma} P \quad \Gamma \vdash_{\sigma} Q}{\Gamma \vdash_{\sigma} P \mid Q}$$

(RES)

$$\frac{\Gamma, b : \theta \vdash_{\sigma} P}{\Gamma \vdash_{\sigma} (\nu b)P}$$

(SUB)

$$\frac{\Gamma \vdash_{\sigma} P \quad \sigma' \leq \sigma}{\Gamma \vdash_{\sigma'} P}$$

(REC<sub>1</sub>)

$$\frac{\Gamma(A) = \sigma}{\Gamma \vdash_{\sigma} A(\tilde{e})}$$

(REC<sub>2</sub>)

$$\frac{\Gamma, A : \sigma \vdash_{\sigma} P}{\Gamma \vdash_{\sigma} (\mathbf{rec} A(\tilde{x}).P)(\tilde{e})}$$

## Soundness of the type system for PBNDC

**Lemma** [ $\approx_{\mathcal{H}}$  – invariance under high actions]

If  $\Gamma \vdash_{\sigma} P$  and  $\mathcal{H} = \{ a \in \mathcal{N} : \Gamma(a) = h \}$ . If  $P \xrightarrow{\alpha} P'$  and  $\Gamma(\alpha) = h$  then  $P \approx_{\mathcal{H}} P'$ .

Main result: **typability**  $\Rightarrow$  **persistent security** (PBNDC):

**Theorem** [Soundness]

If  $\Gamma \vdash_{\sigma} P$  then  $P \approx_{\mathcal{H}} (\nu\mathcal{H})P$ , where  $\mathcal{H} = \{ a \in \mathcal{N} : \Gamma(a) = h \}$ .

## Milner's translation of PARIMP into CCS (1/4)

A **variable**  $X$  is modelled by a *register*:

$$Reg_X(v) \stackrel{\text{def}}{=} put_X(x).Reg_X(x) + \overline{get_X}\langle v \rangle.Reg_X(v)$$

A **state**  $s$  is mapped to a *pool of registers*:

$$[[s]] = Reg_{X_1}(s(X_1)) \mid \cdots \mid Reg_{X_n}(s(X_n)) \quad \text{if } \text{dom}(s) = \{X_1, \dots, X_n\}$$

An **expression**  $E = F(X_1, \dots, X_n)$  is mapped to:

$$[[F(X_1, \dots, X_n)]] = get_{X_1}(x_1) \cdots get_{X_n}(x_n). \overline{\text{res}}\langle f(x_1, \dots, x_n) \rangle. \mathbf{0}$$

Auxiliary operator *Into*, for transmission of values:

$$P \text{ Into}(x) Q \stackrel{\text{def}}{=} (P \mid \text{res}(x).Q) \setminus \text{res}$$

## Translation of PARIMP into CCS (2/4)

A special channel **done**, on which processes signal **termination**.

Auxiliary operators *Done*, *Before* and *Par*:

$$Done \stackrel{\text{def}}{=} \overline{\text{done}}. \mathbf{0}$$

$$C \text{ Before } D \stackrel{\text{def}}{=} (C[d/\text{done}] \mid d.D) \setminus d$$

$$C_1 \text{ Par } C_2 \stackrel{\text{def}}{=} ((C_1[d_1/\text{done}] \mid C_2[d_2/\text{done}]) \mid \\ (d_1.d_2.Done + d_2.d_1.Done)) \setminus \{d_1, d_2\}$$

## Translation of PARIMP into CCS (3/4)

Translation of **commands**:

$$\llbracket \text{nil} \rrbracket = \textit{Done}$$

$$\llbracket X := E \rrbracket = \llbracket E \rrbracket \textit{Into}(x) (\overline{\textit{put}_X} \langle x \rangle. \textit{Done})$$

$$\llbracket C ; D \rrbracket = \llbracket C \rrbracket \textit{Before} \llbracket D \rrbracket$$

$$\llbracket (\textit{if } E \textit{ then } C_1 \textit{ else } C_2) \rrbracket = \llbracket E \rrbracket \textit{Into}(x) (\textit{if } x \textit{ then } \llbracket C_1 \rrbracket \textit{ else } \llbracket C_2 \rrbracket)$$

$$\begin{aligned} \llbracket (\textit{while } E \textit{ do } C) \rrbracket &= W, \text{ where } W \stackrel{\textit{def}}{=} \llbracket E \rrbracket \textit{Into}(x) \\ &\quad (\textit{if } x \textit{ then } \llbracket C \rrbracket \textit{Before } W \textit{ else } \textit{Done}) \end{aligned}$$

$$\llbracket (C_1 \parallel C_2) \rrbracket = \llbracket C_1 \rrbracket \textit{Par} \llbracket C_2 \rrbracket$$

## Translation of PARIMP into CCS (4/4)

Translation of **configurations**  $\langle C, s \rangle$ :

$$\llbracket \langle C, s \rangle \rrbracket = ( \llbracket C \rrbracket \mid \llbracket s \rrbracket ) \setminus Acc_s \cup \{\mathbf{done}\}$$

where  $Acc_s$  is the *access sort* of state  $s$ :

$$Acc_s \stackrel{\text{def}}{=} \{ get_X, put_X \mid X \in \text{dom}(s) \}$$

Problem: **atomicity** of assignments is **not preserved** !

$$C = (X := X + 1 \parallel X := X + 1)$$

## Problem with atomicity (1/2)

Program  $C = (X := X + 1 \parallel X := X + 1)$

The translation of  $C$  is:

$$\begin{aligned} \llbracket C \rrbracket &= ((get_X(x). \overline{\mathbf{res}}\langle x + 1 \rangle \mid \mathbf{res}(y). \overline{put_X}\langle y \rangle. \overline{d_1}) \setminus \mathbf{res} \\ &\mid (get_X(x). \overline{\mathbf{res}}\langle x + 1 \rangle \mid \mathbf{res}(y). \overline{put_X}\langle y \rangle. \overline{d_2}) \setminus \mathbf{res} \\ &\mid (d_1. d_2. Done + d_2. d_1. Done)) \setminus \{d_1, d_2\} \end{aligned}$$

The second  $get_X$  action may be executed before the first  $\overline{put_X}$

$\Rightarrow$  the same value  $v_1 = v_0 + 1$  may be assigned twice to  $X$ .

## Problem with atomicity (2/2)

Suppose  $X$  has low level:

$$C_L = (X_L := X_L + 1 \parallel X_L := X_L + 1)$$

Consider the interleaving of the assignments in  $C_L$ :

$$D_L = (X_L := X_L + 1; X_L := X_L + 1)$$

Security is not preserved:

$\hat{C} = (\text{if } z_H = 0 \text{ then } C_L \text{ else } D_L)$  is secure, but  $\llbracket \hat{C} \rrbracket$  is not secure.

## Adapting the translation (1/2)

A **global semaphore** to ensure atomicity:

$$Sem \stackrel{\text{def}}{=} \text{lock. unlock. Sem}$$

Adapted translation of assignments and configurations:

$$\llbracket X := E \rrbracket = \overline{\text{lock}}. \llbracket E \rrbracket \text{ Into } (x) (\overline{\text{put}_X} \langle x \rangle. \overline{\text{unlock}}. \text{Done})$$

$$\llbracket \langle C, s \rangle \rrbracket = (\llbracket C \rrbracket \mid \llbracket s \rrbracket \mid Sem) \setminus Acc_s \cup \{\text{done, lock, unlock}\}$$

**Atomic translation** of expression  $E$ :

$$\llbracket F(X_1, \dots, X_n) \rrbracket_{at} = \overline{\text{lock}}. \text{getseq}_{\tilde{X}}(\tilde{x}). \overline{\text{res}} \langle f(\tilde{x}) \rangle. \overline{\text{unlock}}. \mathbf{0}$$

## Adapting the translation (2/2)

Adapted translation of conditionals and loops:

$$\llbracket(\text{if } E \text{ then } C_1 \text{ else } C_2)\rrbracket = \llbracket E \rrbracket_{at} \text{Into}(x) (\text{if } x \text{ then } \llbracket C_1 \rrbracket \text{ else } \llbracket C_2 \rrbracket)$$

$$\llbracket(\text{while } E \text{ do } C)\rrbracket = W, \text{ where } W \stackrel{\text{def}}{=} \llbracket E \rrbracket_{at} \text{Into}(x) \\ (\text{if } x \text{ then } \llbracket C \rrbracket \text{ Before } W \text{ else } \text{Done})$$

## Security is preserved by the translation

To set an **operational correspondence** between  $\langle C, s \rangle$  and its image:

$$\llbracket \langle C, s \rangle \rrbracket = (\llbracket C \rrbracket \mid \llbracket s \rrbracket \mid Sem) \setminus Acc_s \cup \{\text{done, lock, unlock}\}$$

one needs a means to observe changes performed by  $\llbracket C \rrbracket$  on  $\llbracket s \rrbracket$ .

Observable register  $OReg_X$ :

$$\begin{aligned} OReg_X(v) \stackrel{\text{def}}{=} & \text{put}_X(x).OReg_X(x) + \overline{\text{get}_X}\langle v \rangle.OReg_X(v) + \\ & \overline{\text{lock}}.(\text{in}_X(x).\overline{\text{unlock}}.OReg_X(x) + \overline{\text{unlock}}.OReg_X(x)) + \\ & \overline{\text{lock}}.(\overline{\text{out}_X}\langle v \rangle.\overline{\text{unlock}}.OReg_X(v) + \overline{\text{unlock}}.OReg_X(v)) \end{aligned}$$

## Operational correspondence

Labelled transitions  $\xrightarrow{in_X v}$  and  $\xrightarrow{\overline{out}_X}$  (and  $\xrightarrow{\tau} \stackrel{\text{def}}{=} \rightarrow$ ) for configurations:

$$\begin{array}{c} \text{(IN-OP)} \quad \frac{X \in \text{dom}(s)}{\langle C, s \rangle \xrightarrow{in_X v} \langle C, s[v/X] \rangle} \qquad \text{(OUT-OP)} \quad \frac{s(X) = v}{\langle C, s \rangle \xrightarrow{\overline{out}_X v} \langle C, s \rangle} \end{array}$$

Transitions are preserved and reflected by the translation:

1.  $\langle C, s \rangle \xrightarrow{\alpha} \langle C', s' \rangle$  implies  $\exists P. \llbracket \langle C, s \rangle \rrbracket \xrightarrow{\alpha} P \approx \llbracket \langle C', s' \rangle \rrbracket$
2.  $\llbracket \langle C, s \rangle \rrbracket \xrightarrow{\alpha} P$  implies either  $P \approx \llbracket \langle C, s \rangle \rrbracket$  or  $\exists C', s'. P \approx \llbracket \langle C', s' \rangle \rrbracket \wedge \langle C, s \rangle \xrightarrow{\hat{\alpha}} \langle C', s' \rangle$ .

**Security is preserved:**  $C$  secure  $\Rightarrow \llbracket \langle C, s \rangle \rrbracket$  satisfies PBNDC.

## Types are not preserved by the translation

Consider the program, typable in PARIMP:

$$C = (X_H := X_H + 1; Y_L := Y_L + 1)$$

Translation of  $C$ :

$$(\nu d) \left( \overline{\text{lock}}. (\nu \text{res}_1) (get_{X_H}(x). \overline{\text{res}_1}\langle x + 1 \rangle \mid \text{res}_1(z_1). \overline{\text{put}_{X_H}}\langle z_1 \rangle. \overline{\text{unlock}}. \overline{d}) \mid \right. \\ \left. d. \overline{\text{lock}}. (\nu \text{res}_2) (get_{Y_L}(y). \overline{\text{res}_2}\langle y + 1 \rangle \mid \text{res}_2(z_2). \overline{\text{put}_{Y_L}}\langle z_2 \rangle. \overline{\text{unlock}}. \overline{\text{done}}) \right)$$

Which **choice of security levels** for channels `lock`, `unlock` and `d`?

## Adapting the type system

Idea: restricted high actions without parameters do not leak information if they are **granted to be enabled uniformly** in all low-equivalent states.

True for actions `lock, unlock`: **the semaphore is always released** after a finite number of steps.

Instead, action `done` may be prevented by **deadlock** or **divergence**:

⇒ by restricting the use of loops in the source program one may obtain an ad hoc solution.

# Conclusion

1. **Security preserving translation** into CCS (variant of Milner's), extending work by Focardi, Rossi and Sabelfeld '05 in two ways:
  - **parallel** imperative language
  - **time insensitive** security property
2. **Equivalence preserving translation** (as a by-product).
3. **Security type system for CCS** (PBNDC), inspired by Pottier '02, which needs to be tuned to reflect a type system on PARIMP.

Future work:

- more general security type system for CCS
- move to more complex languages