

# Termination for Concurrent Languages

Romain Demangeon  
ENS Lyon - Università di Bologna

PhD defense  
ENS Lyon, November the 23rd 2010

Termination and Concurrency

The Framework:  $\pi$ -calculus

Type Systems for Termination

Termination in impure languages

Conclusion

# Termination and Concurrency

The Framework:  $\pi$ -calculus

Type Systems for Termination

Termination in impure languages

Conclusion

# Formal Methods

## Verifying programs and protocols

- ▶ **Sequential**. Programming computers, chips.
- ▶ **Mobile/Distributed**. Programs running on planes, mobile phones, web applications.
- ▶ Risks  $\rightsquigarrow$  Need for verification:
  - ▶ Soundness for avionics.
  - ▶ Security for bank account management.
  - ▶ Predictability for web applications.

# Formal Methods

## Verifying programs and protocols

- ▶ **Sequential.** Programming computers, chips.
- ▶ **Mobile/Distributed.** Programs running on planes, mobile phones, web applications.
- ▶ Risks  $\rightsquigarrow$  Need for verification:
  - ▶ Soundness for avionics.
  - ▶ Security for bank account management.
  - ▶ Predictability for web applications.

## Abstracting

- ▶ Working in an abstract and mathematical framework.
- ▶ Extracting the essence of programs.
- ▶ Designing static analyses on abstractions.

# Concurrency Theory

## Motivations

- ▶ Describing programs running:
  - ▶ on a multi-unit platform (multicore, FPGA, GPU)
  - ▶ on several heterogeneous units.
- ▶ Describing network protocols (Web Services).

## Abstracting distributed and mobile protocols

- ▶ Non-determinism, competition for resources.
- ▶ Evolution of the shape of systems.
- ▶ Shared memory / Message Passing.

# Message-passing framework

## Entities in a concurrent world

- ▶ Servers, always available:

$\mathcal{S} : s(\text{arg}) \triangleright \boxed{\text{body of } \mathcal{S}}$

- ▶ Clients, sending requests to servers, waiting for answers:

$\mathcal{C} : \text{call}\langle s, \text{request} \rangle; \text{receive}\langle \text{ans} \rangle; \text{call}\langle s', \text{ans} \rangle.$

# Message-passing framework

## Entities in a concurrent world

- ▶ Servers, always available:

$\mathcal{S} : s(\text{arg}) \triangleright \boxed{\text{body of } \mathcal{S}}$

- ▶ Clients, sending requests to servers, waiting for answers:

$\mathcal{C} : \text{call}\langle s, \text{request} \rangle; \text{receive}\langle \text{ans} \rangle; \text{call}\langle s', \text{ans} \rangle.$

- ▶ Programs, several clients and servers:

$\mathcal{S}_1 : s_1(\text{arg}_1) \triangleright \dots \quad \mathcal{S}'_1 : s_1(\text{arg}_1) \triangleright \text{call}\langle s_2, \text{arg}_1 \rangle$

$\mathcal{S}_2 : s_2(\text{arg}_2) \triangleright \dots$

$\mathcal{C}_1 : \text{call}\langle s_1, \text{request}_1 \rangle \dots \quad \mathcal{C}_2 : \text{call}\langle s_1, \text{request}_2 \rangle \dots$

# Abstracting servers

- ▶ A request may induce other requests:

$$\mathcal{S}_f : \text{fibonacci}(\text{arg}) \triangleright \dots \text{call}\langle \text{fibonacci}, \dots \rangle \dots$$
$$\text{call}\langle \text{fibonacci}, \dots \rangle \dots \text{call}\langle \text{sum}, \dots \rangle \dots$$

- ▶ Abstracting request calls:  $\mathcal{S}_f : \text{fibonacci} \triangleright (\text{fibonacci} \mid \text{fibonacci} \mid \text{sum})$ .

# Unsafe behaviours

## Loop between servers

$$S_1 : s_1(\text{arg}_1) \triangleright \text{call}\langle s_2, \text{arg}_1 \rangle$$
$$S_2 : s_2(\text{arg}_2) \triangleright \text{call}\langle s_1, \text{arg}_2 \rangle$$
$$C : \text{call}\langle s_1, \text{Arctan}(8) \rangle; \text{receive}(\text{ans})$$

- ▶  $s_1$  calls  $s_2$  which calls  $s_1, \dots$
- ▶  $C$  waits forever.
- ▶ Abstracting:  $\{s_1 \triangleright s_2, s_2 \triangleright s_1\}$ .

# Termination

## Definition

A program **terminates** when all its executions are finite.

## Non terminating programs

- ▶  $(\lambda x.x x) (\lambda y.y y)$ .
- ▶ while **true** do skip.
- ▶  $\{[o] \rightarrow [oo], [o] \rightarrow []\}$ ,

## Termination in programming theory

- ▶ for functional programs (type systems, realisability,  $\lambda_{ST}$ ),
- ▶ for imperative programs (abstract interpretation, loop analyses),
- ▶ for rewriting systems (rewriting orders, polynomial interpretations).

# Simply-typed $\lambda$ -calculus

- ▶ Abstracting functional programs.
- ▶ Simple types ( $A \rightarrow B$ ).
- ▶ Different proofs of termination.
- ▶ Can be enriched (polymorphism, sum types, ...).

# Termination in concurrency theory

## A desirable property

- ▶ Servers run forever,  
**but** answers have to be produced eventually.
- ▶ Prerequisite for other properties: lock-freedom, complexity bounds, fairness.

## A challenging analysis

- ▶ Evolving topology, loose structures, non-determinism.
- ▶ Few existing results:
  - ▶ Terminator [CookPodelskiRybalchenko07], termination of threads.

# Back to the example

## A first idea

- ▶ Enumerating the servers  $(s_i)_{1 \leq i \leq n}$ .
- ▶ Restricting requests between servers:
  - ▶  $S$  can only make one call
  - ▶  $S : s_i(\dots)$  can only send a request to  $s_j$  if  $j < i$ .

# Back to the example

## A first idea

- ▶ Enumerating the servers  $(s_i)_{1 \leq i \leq n}$ .
- ▶ Restricting requests between servers:
  - ▶  $S$  can only make one call
  - ▶  $S : s_i(\dots)$  can only send a request to  $s_j$  if  $j < i$ .
- ▶ Abstraction:  $(s_i \triangleright s_{l_i})_{1 \leq i \leq n}$   
with  $l_i < i$
- ▶ For each client request  $s_{k_1}$ :  
 $s_{k_1} \rightsquigarrow s_{k_2} \rightsquigarrow \dots$  with  $k_{i+1} = k_{l_i} < k_i$ .
- ▶ Well-foundedness of  $\mathbb{N} \Rightarrow$  termination.

# Back to the example

## A first idea

- ▶ Enumerating the servers  $(s_i)_{1 \leq i \leq n}$ .
- ▶ Restricting requests between servers:
  - ▶  $S$  can only make one call
  - ▶  $S : s_i(\dots)$  can only send a request to  $s_j$  if  $j < i$ .
- ▶ Abstraction:  $(s_i \triangleright s_{l_i})_{1 \leq i \leq n}$   
with  $l_i < i$
- ▶ For each client request  $s_{k_1}$ :  
 $s_{k_1} \rightsquigarrow s_{k_2} \rightsquigarrow \dots$  with  $k_{i+1} = k_{l_i} < k_i$ .
- ▶ Well-foundedness of  $\mathbb{N} \Rightarrow$  termination.
- ▶ Limited **expressiveness**.

# Two main techniques for terminations

## Using weights

- ▶ Assigning levels to requests.
- ▶ Defining a notion of weight for a program.
- ▶ Ensuring that the weight decreases at each call.
- ▶ [DengSangiorgi06]

## Using logical relations

- ▶ Widely used technique (sequential setting).
- ▶ Restricting the definitions of servers.
- ▶ Assigning types to these servers.
- ▶ Using logical relations to ensure termination.
- ▶ [YoshidaBergerHonda04] and [Sangiorgi06].

Termination and Concurrency

The Framework:  $\pi$ -calculus

Type Systems for Termination

Termination in impure languages

Conclusion

# Formalism: a simple $\pi$ -calculus

## Syntax

$P ::= \mathbf{0} \mid P \mid P \mid \bar{a}\langle v \rangle.P \mid a(x).P$

- ▶ Names (or channels)  $a, b, c, v, w, x, y$
- ▶  $P_1 \mid P_2$ : associative; commutative,  $\mathbf{0}$  as neutral element.

# Formalism: a simple $\pi$ -calculus

## Syntax

$P ::= \mathbf{0} \mid P \mid P \mid \bar{a}\langle v \rangle.P \mid a(x).P$

- ▶ Names (or channels)  $a, b, c, v, w, x, y$
- ▶  $P_1 \mid P_2$ : associative; commutative,  $\mathbf{0}$  as neutral element.
- ▶ Restriction  $(\nu a) P$ ,
  - ▶ Relevant to termination.
  - ▶ Involve technical proofs.
  - ▶ Not used in this presentation.

# Formalism: a simple $\pi$ -calculus

## Syntax

$P ::= \mathbf{0} \mid P \mid P \mid \bar{a}\langle v \rangle.P \mid a(x).P$

- ▶ Names (or channels)  $a, b, c, v, w, x, y$
- ▶  $P_1 \mid P_2$ : associative; commutative,  $\mathbf{0}$  as neutral element.
- ▶ Restriction  $(\nu a) P$ ,
  - ▶ Relevant to termination.
  - ▶ Involve technical proofs.
  - ▶ Not used in this presentation.

## Communication rule

$$\text{▶ } \frac{}{P_0 \mid a(x).P_1 \mid \bar{a}\langle v \rangle.P_2 \longrightarrow P_0 \mid P_1\{v/x\} \mid P_2}$$

# An example of reduction

$$\overline{P_0 \mid a(x).P_1 \mid \bar{a}(v).P_2} \longrightarrow P_0 \mid P_1\{v/x\} \mid P_2$$

$$P_1 = \bar{a}(c).0 \mid a(x).\bar{b}(x).0 \mid a(y).y(z).0 \mid \bar{c}(v).0$$

$$a(x).\bar{b}(x).0 \mid c(z).0 \mid \bar{c}(v).0$$

$$a(x).\bar{b}(x).0$$

$$\bar{b}(c).0 \mid a(y).y(z).0 \mid \bar{c}(v).0$$

# An example of reduction

$$\overline{P_0 \mid a(x).P_1 \mid \bar{a}(v).P_2} \longrightarrow P_0 \mid P_1\{v/x\} \mid P_2$$

$$P_1 = \bar{a}(c).0 \mid a(x).\bar{b}(x).0 \mid a(y).y(z).0 \mid \bar{c}(v).0$$

$$a(x).\bar{b}(x).0 \mid c(z).0 \mid \bar{c}(v).0$$

$$a(x).\bar{b}(x).0$$

$$\bar{b}(c).0 \mid a(y).y(z).0 \mid \bar{c}(v).0$$

Remark:

- ▶ Non-confluence.
- ▶ Terminating calculus (2 prefixes consumed at each reduction).

## Adding replication

$P ::= \mathbf{0} \mid P \mid P \mid \bar{a}\langle v \rangle.P \mid a(x).P \mid !a(x).P.$

### Replicated inputs

- ▶ Associated semantics rule:

$$\frac{}{P_0 \mid !a(x).P_1 \mid \bar{a}\langle v \rangle.P_2 \longrightarrow P_0 \mid !a(x).P_1 \mid P_1\{v/x\} \mid P_2}$$

# Adding replication

$$P ::= \mathbf{0} \mid P|P \mid \bar{a}\langle v \rangle.P \mid a(x).P \mid !a(x).P.$$

## Replicated inputs

- ▶ Associated semantics rule:

$$\frac{}{P_0 \mid !a(x).P_1 \mid \bar{a}\langle v \rangle.P_2 \longrightarrow P_0 \mid !a(x).P_1 \mid P_1\{v/x\} \mid P_2}$$

- ▶ Behaviour:
  - ▶ Persistent server.
  - ▶ Receives a request on  $a$ .
  - ▶ Spawns a computation  $P$  and remains available.
- ▶ Server library / Current State:

$$!a(x) \dots \mid !a(y) \dots \mid !b(z) \dots \mid \bar{a}\langle v \rangle.P \mid c(y).Q$$

# Divergence

Remark: in the following, CCS notation:  $!a.\bar{b}$   
instead of  $!a(x).\bar{b}\langle x \rangle.0$

## Diverging processes

- ▶  $D_1 = !a.\bar{a} \mid \bar{a}$
- ▶  $D_2 = !a.\bar{b} \mid !b.\bar{a} \mid \bar{a}$
- ▶  $D_3 = c(x).!a.\bar{x} \mid \bar{a} \mid \bar{c}\langle b \rangle \mid \bar{c}\langle a \rangle$

# Divergence

Remark: in the following, CCS notation:  $!a.\bar{b}$   
instead of  $!a(x).\bar{b}\langle x \rangle.0$

## Diverging processes

- ▶  $D_1 = !a.\bar{a} \mid \bar{a} \longrightarrow D_1$
- ▶  $D_2 = !a.\bar{b} \mid !b.\bar{a} \mid \bar{a}$
- ▶  $D_3 = c(x).!a.\bar{x} \mid \bar{a} \mid \bar{c}\langle b \rangle \mid \bar{c}\langle a \rangle$

# Divergence

Remark: in the following, CCS notation:  $!a.\bar{b}$   
instead of  $!a(x).\bar{b}\langle x \rangle.0$

## Diverging processes

- ▶  $D_1 = !a.\bar{a} \mid \bar{a} \longrightarrow D_1$
- ▶  $D_2 = !a.\bar{b} \mid !b.\bar{a} \mid \bar{a} \longrightarrow \longrightarrow D_2$
- ▶  $D_3 = c(x).!a.\bar{x} \mid \bar{a} \mid \bar{c}\langle b \rangle \mid \bar{c}\langle a \rangle$

# Divergence

Remark: in the following, CCS notation:  $!a.\bar{b}$   
instead of  $!a(x).\bar{b}\langle x \rangle.0$

## Diverging processes

- ▶  $D_1 = !a.\bar{a} \mid \bar{a} \longrightarrow D_1$
- ▶  $D_2 = !a.\bar{b} \mid !b.\bar{a} \mid \bar{a} \longrightarrow \longrightarrow D_2$
- ▶  $D_3 = c(x).!a.\bar{x} \mid \bar{a} \mid \bar{c}\langle b \rangle \mid \bar{c}\langle a \rangle \longrightarrow D_1 \mid \bar{c}\langle b \rangle$

Termination and Concurrency

The Framework:  $\pi$ -calculus

Type Systems for Termination

Termination in impure languages

Conclusion

# Type systems

## Principles

- ▶ Associating types to names:  $a : \#$  ( $\#$  nat)
- ▶  $P$  well-typed:  $\vdash P$
- ▶ Typing rules correspond to the structure of  $P$ :

$$\frac{\vdash P_1 \quad \vdash P_2 \quad \text{some condition on } P_1 \text{ and } P_2}{\vdash P_1 \mid P_2}$$

# Weight-based type system

## Levels and Weights

- ▶ [DengSangiorgi06]
- ▶ Inspired by rewriting orders (well-foundedness).
- ▶ Types give a *level* to each name:

$$c(x).\bar{x}\langle 8 \rangle \rightsquigarrow x : \#^2 \text{ nat}, c : \#^1 (\#^2 \text{ nat})$$

# Weight-based type system

## Levels and Weights

- ▶ [DengSangiorgi06]
- ▶ Inspired by rewriting orders (well-foundedness).
- ▶ Types give a *level* to each name:

$$c(x).\bar{x}\langle 8 \rangle \rightsquigarrow x : \#^2 \text{ nat}, c : \#^1 (\#^2 \text{ nat})$$

## Controlling loops

- ▶ **Weight** of a process,  $\vdash P : n$ : maximum level of an output in the **current state** (unreplicated part).

# Weight-based type system

## Levels and Weights

- ▶ [DengSangiorgi06]
- ▶ Inspired by rewriting orders (well-foundedness).
- ▶ Types give a *level* to each name:

$$c(x).\bar{x}\langle 8 \rangle \rightsquigarrow x : \#^2 \text{ nat}, c : \#^1 (\#^2 \text{ nat})$$

## Controlling loops

- ▶ **Weight** of a process,  $\vdash P : n$ : maximum level of an output in the **current state** (unreplicated part).
- ▶ Control on replications:
  - ▶  $!a^k(x).P$  with  $\vdash P : n$  requires  $k > n$ .
- ▶ Soundness: decreasing measure.

## Example of reduction

- ▶  $T_1 = !a . (\bar{b} \mid \bar{b} \mid \bar{c}) \mid !b . (\bar{c} \mid \bar{c})$
- ▶  $T_1 \mid \bar{a} \mid \bar{b} \rightarrow T_1 \mid \bar{a} \mid \bar{c} \mid \bar{c} \rightarrow T_1 \mid \bar{b} \mid \bar{b} \mid \bar{c} \mid \bar{c} \mid \bar{c} \rightarrow \rightarrow \not\rightarrow$

## Example of reduction

- ▶  $T_1 = !a^3.(\bar{b}^2 \mid \bar{b}^2 \mid \bar{c}^1) \mid !b^2.(\bar{c}^1 \mid \bar{c}^1)$
- ▶  $T_1 \mid \bar{a} \mid \bar{b} \rightarrow_2 T_1 \mid \bar{a} \mid \bar{c} \mid \bar{c} \rightarrow_3 T_1 \mid \bar{b} \mid \bar{b} \mid \bar{c} \mid \bar{c} \mid \bar{c} \rightarrow_2 \rightarrow_2 \not\rightarrow$
- ▶  $\{3, 2\} \rightarrow_2 \{3, 1, 1\} \rightarrow_3 \{2, 2, 1, 1, 1\} \rightarrow_2 \{2, 1, 1, 1, 1, 1\} \rightarrow_2 \{1, 1, 1, 1, 1, 1, 1\}$

## Example of reduction

- ▶  $T_1 = !a^3.(\bar{b}^2 \mid \bar{b}^2 \mid \bar{c}^1) \mid !b^2.(\bar{c}^1 \mid \bar{c}^1)$
- ▶  $T_1 \mid \bar{a} \mid \bar{b} \rightarrow_2 T_1 \mid \bar{a} \mid \bar{c} \mid \bar{c} \rightarrow_3 T_1 \mid \bar{b} \mid \bar{b} \mid \bar{c} \mid \bar{c} \mid \bar{c} \rightarrow_2 \rightarrow_2 \not\rightarrow$
- ▶  $\{3, 2\} \rightarrow_2 \{3, 1, 1\} \rightarrow_3 \{2, 2, 1, 1, 1\} \rightarrow_2 \{2, 1, 1, 1, 1, 1\} \rightarrow_2 \{1, 1, 1, 1, 1, 1, 1\}$

## The diverging examples are rejected

- ▶  $D_1 = !a^n.\bar{a}^n \mid \bar{a}^n$
- ▶  $D_2 = !a^n.\bar{b}^k \mid !b^k.\bar{a}^n \mid \bar{a}^n$
- ▶  $D_3 = c^t(x).!a^n.\bar{x}^k \mid \bar{a}^n \mid \bar{c}^t\langle a \rangle \mid \bar{c}^t\langle b \rangle.$

## Example of reduction

- ▶  $T_1 = !a^3.(\bar{b}^2 \mid \bar{b}^2 \mid \bar{c}^1) \mid !b^2.(\bar{c}^1 \mid \bar{c}^1)$
- ▶  $T_1 \mid \bar{a} \mid \bar{b} \rightarrow_2 T_1 \mid \bar{a} \mid \bar{c} \mid \bar{c} \rightarrow_3 T_1 \mid \bar{b} \mid \bar{b} \mid \bar{c} \mid \bar{c} \mid \bar{c} \rightarrow_2 \rightarrow_2 \not\rightarrow$
- ▶  $\{3, 2\} \rightarrow_2 \{3, 1, 1\} \rightarrow_3 \{2, 2, 1, 1, 1\} \rightarrow_2 \{2, 1, 1, 1, 1, 1\} \rightarrow_2 \{1, 1, 1, 1, 1, 1, 1\}$

## The diverging examples are rejected

- ▶  $D_1 = !a^n.\bar{a}^n \mid \bar{a}^n$  not typable:  $n > n$
- ▶  $D_2 = !a^n.\bar{b}^k \mid !b^k.\bar{a}^n \mid \bar{a}^n$
- ▶  $D_3 = c^t(x).!a^n.\bar{x}^k \mid \bar{a}^n \mid \bar{c}^t\langle a \rangle \mid \bar{c}^t\langle b \rangle$ .

## Example of reduction

- ▶  $T_1 = !a^3.(\bar{b}^2 \mid \bar{b}^2 \mid \bar{c}^1) \mid !b^2.(\bar{c}^1 \mid \bar{c}^1)$
- ▶  $T_1 \mid \bar{a} \mid \bar{b} \rightarrow_2 T_1 \mid \bar{a} \mid \bar{c} \mid \bar{c} \rightarrow_3 T_1 \mid \bar{b} \mid \bar{b} \mid \bar{c} \mid \bar{c} \mid \bar{c} \rightarrow_2 \rightarrow_2 \not\rightarrow$
- ▶  $\{3, 2\} \rightarrow_2 \{3, 1, 1\} \rightarrow_3 \{2, 2, 1, 1, 1\} \rightarrow_2 \{2, 1, 1, 1, 1, 1\} \rightarrow_2 \{1, 1, 1, 1, 1, 1, 1\}$

## The diverging examples are rejected

- ▶  $D_1 = !a^n.\bar{a}^n \mid \bar{a}^n$  not typable:  $n > n$
- ▶  $D_2 = !a^n.\bar{b}^k \mid !b^k.\bar{a}^n \mid \bar{a}^n$  not typable:  $n > k > n$ .
- ▶  $D_3 = c^t(x).!a^n.\bar{x}^k \mid \bar{a}^n \mid \bar{c}^t\langle a \rangle \mid \bar{c}^t\langle b \rangle$ .

## Example of reduction

- ▶  $T_1 = !a^3.(\bar{b}^2 \mid \bar{b}^2 \mid \bar{c}^1) \mid !b^2.(\bar{c}^1 \mid \bar{c}^1)$
- ▶  $T_1 \mid \bar{a} \mid \bar{b} \rightarrow_2 T_1 \mid \bar{a} \mid \bar{c} \mid \bar{c} \rightarrow_3 T_1 \mid \bar{b} \mid \bar{b} \mid \bar{c} \mid \bar{c} \mid \bar{c} \rightarrow_2 \rightarrow_2 \not\rightarrow$
- ▶  $\{3, 2\} \rightarrow_2 \{3, 1, 1\} \rightarrow_3 \{2, 2, 1, 1, 1\} \rightarrow_2 \{2, 1, 1, 1, 1, 1\} \rightarrow_2 \{1, 1, 1, 1, 1, 1, 1\}$

## The diverging examples are rejected

- ▶  $D_1 = !a^n.\bar{a}^n \mid \bar{a}^n$  not typable:  $n > n$
- ▶  $D_2 = !a^n.\bar{b}^k \mid !b^k.\bar{a}^n \mid \bar{a}^n$  not typable:  $n > k > n$ .
- ▶  $D_3 = c^t(x).!a^n.\bar{x}^k \mid \bar{a}^n \mid \bar{c}^t\langle a \rangle \mid \bar{c}^t\langle b \rangle$ .  
 $c : \#^t (\#^n T)$  thus  $n = k$  and  $n > k$ .

# Expressiveness of weight-based techniques

## Increasing expressiveness

- ▶ Goal: recognising more terminating behaviours.
- ▶ Refining the technique.
  - ▶ Allowing controlled recursion  $!a(x).(\dots \bar{a}\langle v \rangle \dots)$ .

## Limitations

- ▶ Too fast a termination:  
at most  $N^{N+1}$  steps for  $P$  of size  $N$ .
- ▶ Not able to type the standard encoding of  $\lambda_{\mathbf{ST}}$  into  $\pi$  [SangiorgiWalker01].
  - ▶ Existence of a counter example  $M_0 \in \lambda_{\mathbf{ST}}$  s.t.  $\llbracket M_0 \rrbracket_\rho$  not typable using any weight-based type system.

# A semantical method

## Principles [Sangiorgi06]

- ▶ Using the encoding of  $\lambda_{ST}$  into  $\pi$ .
- ▶ Termination of a subset of  $\pi$  by logical relations:
  - ▶ Assigning types to processes. ( $\vdash P : T$ )
  - ▶ Defining the interpretation of a type as a set of terminating processes. ( $P \in [T]$ )
  - ▶ Proving that every typable process belongs to the interpretation of its type. ( $\vdash P : T$  implies  $P \in [T]$ )

## Limitations

- ▶ Strong constraints on processes:
  - ▶ Only one input for each name, replicated and implementing an immutable server.
- ▶ Yields a confluent calculus.
- ▶ Limited “concurrent expressiveness”.

# Combining the two methods

- ▶ Goal: termination for concurrent programs containing both
  - ▶ typically functional procedures
  - ▶ and non-trivial concurrent behaviour.
- ▶ Main difficulty: the two parts can collaborate to create divergence.
- ▶ Exploiting the two proof techniques:
  - ▶ logical relations
  - ▶ and the weights.

Termination and Concurrency

The Framework:  $\pi$ -calculus

Type Systems for Termination

Termination in impure languages

Conclusion

# General idea

- ▶ Starting with a subset of the  $\pi$ -calculus, *terminating* and *functional* .
- ▶ Adding *imperative* names controlled by levels.
  - ▶ *imperative* replications.
  - ▶ *functional* replications.
  - ▶ impure: *functional* + *imperative*.
- ▶ Extending the type system with levels to the whole calculus.
  - ▶ Looser constraints for functional names.
- ▶ Termination proof via a simulation.

# An impure $\pi$ -calculus

$P ::= \mathbf{0} \mid P \mid P \mid \bar{v}(w).P \mid (\nu a) P \mid a(x).P \mid !a(x).P \mid \text{def } f = (x).P \text{ in } P$

## Functional names

- ▶ Some names are **functional**:
  - ▶ Inputs replaced by *definitions*:  $\text{def } f = (x).P_1 \text{ in } P_2$ 
    - ▶ syntactic sugar for  $(\nu f) (!f(x).P_1 \mid P_2)$
    - ▶ no “recursion”:  $\bar{f} \notin \text{fn}(P_1)$ .
    - ▶ no input on  $f$  in  $P_1$  or  $P_2$ .

# An impure $\pi$ -calculus

$P ::= \mathbf{0} \mid P \mid P \mid \bar{v}\langle w \rangle.P \mid (\nu a) P \mid a(x).P \mid !a(x).P \mid \text{def } f = (x).P \text{ in } P$

## Functional names

- ▶ Some names are **functional**:
  - ▶ Inputs replaced by *definitions*:  $\text{def } f = (x).P_1 \text{ in } P_2$ 
    - ▶ syntactic sugar for  $(\nu f) (!f(x).P_1 \mid P_2)$
    - ▶ no “recursion”:  $\bar{f} \notin \text{fn}(P_1)$ .
    - ▶ no input on  $f$  in  $P_1$  or  $P_2$ .
- ▶ Reduction

$$\text{def } f = (x).P_1 \text{ in } (\bar{f}\langle v \rangle.P_2 \mid P_3)$$
$$\longrightarrow \text{def } f = (x).P_1 \text{ in } (P_1\{v/x\} \mid P_2 \mid P_3)$$

# An impure $\pi$ -calculus

$P ::= \mathbf{0} \mid P \mid P \mid \bar{v}\langle w \rangle.P \mid (\nu a) P \mid a(x).P \mid !a(x).P \mid \text{def } f = (x).P \text{ in } P$

## Functional names

- ▶ Some names are **functional**:
  - ▶ Inputs replaced by *definitions*:  $\text{def } f = (x).P_1 \text{ in } P_2$ 
    - ▶ syntactic sugar for  $(\nu f) (!f(x).P_1 \mid P_2)$
    - ▶ no “recursion”:  $\bar{f} \notin \text{fn}(P_1)$ .
    - ▶ no input on  $f$  in  $P_1$  or  $P_2$ .
- ▶ Reduction

$$\text{def } f = (x).P_1 \text{ in } (\bar{f}\langle v \rangle.P_2 \mid P_3)$$
$$\longrightarrow \text{def } f = (x).P_1 \text{ in } (P_1\{v/x\} \mid P_2 \mid P_3)$$

## Imperative names

- ▶ Standard inputs and outputs:  $a(x).P, !a(x).P, \bar{a}\langle v \rangle.P$

## Type system: Imperative inputs

$$\frac{\vdash P : n \quad a : \#_I^k T \quad x : T \quad k > n}{\vdash !a(x).P : 0}$$

$$\frac{\vdash P : n \quad a : \#_I^k T \quad x : T \quad k > n}{\vdash a(x).P : 0}$$

## Type system: Functional definitions

$$\frac{f : \#_{\mathbb{F}}^k T \quad \vdash P_1 : n \quad \vdash P_2 : n' \quad x : T \quad f \notin \text{fn}(P_1) \quad k \geq n}{\vdash \text{def } f = (x).P_1 \text{ in } P_2 : n'}$$

- ▶ Consequence of  $\geq$  on expressiveness: encoding of  $\lambda_{ST}$  in impure  $\pi$  typed with all levels set to 0

# Soundness proof

- ▶ **Theorem:** all typable processes terminate.
- ▶ By absurd:  
consider an infinite derivation from a well-typed process  $P_0$ :

$$P_0 \xrightarrow{2} \xrightarrow{5} \xrightarrow{4} \xrightarrow{5} \xrightarrow{4} \xrightarrow{3} \xrightarrow{5} \dots$$

- ▶ There exists  $p$  maximal such that infinitely often  $\xrightarrow{p}$  or  $\xrightarrow{p}$ .

# Soundness proof

- ▶ **Theorem:** all typable processes terminate.
- ▶ By absurd:  
consider an infinite derivation from a well-typed process  $P_0$ :

$$P_0 \xrightarrow{2} \xrightarrow{5} \xrightarrow{4} \xrightarrow{5} \xrightarrow{4} \xrightarrow{3} \xrightarrow{5} \dots$$

- ▶ There exists  $p$  maximal such that infinitely often  $\xrightarrow{p}$  or  $\xrightarrow{p}$ .
- ▶ The type system enforces that  $\xrightarrow{p}$  appears infinitely often.  
( $\xrightarrow{p}$  involves a strict decreasing)
- ▶ Infinitely many **functional** reductions at level  $p$ .  
Contradiction ? ( $\xrightarrow{\quad} \xrightarrow{\quad} \xrightarrow{\quad} \dots$ )

# Proof by pruning

## Pruning and reduction sequences

$$\begin{array}{ccccccc} \longrightarrow_2 & \longrightarrow_5 & \longrightarrow_4 & \longrightarrow_5 & \longrightarrow_4 & \longrightarrow_3 & \longrightarrow_5 \dots \\ \simeq & \simeq & \simeq & \downarrow \text{pr}^5 & \simeq & \simeq & \longrightarrow \dots \end{array}$$

## Proof sketch

- ▶ Pruning  $\text{pr}^P()$  of impure into pure functional processes.
  - ▶ Removing all imperative parts and some functional parts.

# Proof by pruning

## Pruning and reduction sequences

$$\begin{array}{cccccccc} \longrightarrow_2 & \longrightarrow_5 & \longrightarrow_4 & \longrightarrow_5 & \longrightarrow_4 & \longrightarrow_3 & \longrightarrow_5 & \dots \\ & & & \downarrow \text{pr}^5 & & & & \\ \simeq & \simeq & \simeq & \longrightarrow & \simeq & \simeq & \longrightarrow & \dots \end{array}$$

## Proof sketch

- ▶ **Pruning**  $\text{pr}^P()$  of impure into **pure functional processes**.
  - ▶ Removing all imperative parts and some functional parts.
- ▶ **Simulation lemma**: If  $P \longrightarrow P'$  then:
  1. either  $\text{pr}^P(P) \simeq \text{pr}^P(P')$
  2. or  $\text{pr}^P(P) \longrightarrow \text{pr}^P(P')$ .
  3.  $\longrightarrow^P$  transformed by  $\text{pr}^P$  into  $\longrightarrow$ .

# Proof by pruning

## Pruning and reduction sequences

$$\begin{array}{cccccccc} \longrightarrow_2 & \longrightarrow_5 & \longrightarrow_4 & \longrightarrow_5 & \longrightarrow_4 & \longrightarrow_3 & \longrightarrow_5 & \dots \\ & & & \downarrow \text{pr}^5 & & & & \\ \simeq & \simeq & \simeq & \longrightarrow & \simeq & \simeq & \longrightarrow & \dots \end{array}$$

## Proof sketch

- ▶ **Pruning**  $\text{pr}^P()$  of impure into **pure functional processes**.
  - ▶ Removing all imperative parts and some functional parts.
- ▶ **Simulation lemma**: If  $P \longrightarrow P'$  then:
  1. either  $\text{pr}^P(P) \simeq \text{pr}^P(P')$
  2. or  $\text{pr}^P(P) \longrightarrow \text{pr}^P(P')$ .
  3.  $\longrightarrow^P$  transformed by  $\text{pr}^P$  into  $\longrightarrow$ .
- ▶ Yields  $\longrightarrow \longrightarrow \longrightarrow \dots$ 
  - ▶ Contradiction with termination of the pure functional calculus.

## Definition of pruning

$$\text{pr}^p(a(x).P) = \text{pr}^p(!a(x).P) = \mathbf{0} \qquad \text{pr}^p(\bar{a}\langle v \rangle.P) = \text{pr}^p(P)$$

$$\text{pr}^p(\text{def } f^n = (x).P_1 \text{ in } P_2) = \begin{cases} \text{def } f = (x).\text{pr}^p(P_1) \text{ in } \text{pr}^p(P_2) & \text{if } n = p \\ \text{pr}^p(P_2) & \text{otherwise} \end{cases}$$

$$\text{pr}^p(\bar{f}^n\langle v \rangle.P) = \begin{cases} \bar{f}^n\langle v \rangle.\text{pr}^p(P) & \text{if } n = p \\ P & \text{otherwise} \end{cases}$$

- ▶ Pruning done *at level p*.
- ▶ Removing all **imperative parts**.
- ▶ Removing **functional parts** at levels  $\neq p$ .

## Typing a complex example

$$\text{Sys}_1 \stackrel{\text{def}}{=} \left( \nu \text{lock}_1, \dots, \text{lock}_k \right. \\ \left. \begin{array}{l} \text{lock}_1 \mid \dots \mid \text{lock}_k \mid \\ \text{def } s = (c, x). \bar{c} \langle \text{enc}[c, x] \rangle \text{ in} \\ \text{def } c_1 = C_1 \text{ in} \\ \dots \\ \text{def } c_{k-1} = C_{k-1} \text{ in} \\ \text{def } c_k = C_k \text{ in} \\ (\bar{s} \langle c_1, \text{msg}_1 \rangle \mid \dots \mid \bar{s} \langle c_1, \text{msg}_n \rangle) \end{array} \right)$$

with:

$$\begin{array}{l} C_i \stackrel{\text{def}}{=} (y_i). \overline{\text{lock}_i}. (\text{lock}_i \mid \bar{s} \langle c_{i+1}, \text{dec}_i[y_i] \rangle) \\ C_k \stackrel{\text{def}}{=} (y_k). \overline{\text{lock}_k}. (\text{lock}_k \mid \bar{d} \langle \text{dec}_k[y_k] \rangle) \end{array}$$

## Typing this program

$\text{lock}_i : \#_I^0 \text{ unit}$ ,  $c_i : \#_F^1 \text{ b}$ ,  $s : \#_F^1 (\#_F^1 \text{ b} \times \text{b})$ ,  $\text{msg}_i : \text{b}$ ,  $d : \#_I^1 \text{ b}$

# Contributions of the thesis

- ▶ (in this talk) Termination in an impure  $\pi$  (CONCUR'10).
- ▶ Strong normalisation in  $\lambda_{\text{ref}}$  (submitted).
  - ▶ Soundness of a well-known type and effect system ([Boudol07], [Amadio09], [AmadioMadet10], [Tranquilli10]) by adapting the previous method.
- ▶ Weight-based systems:
  - ▶ Termination techniques for higher-order concurrent systems (FSEN'09).
    - ▶ Proving termination for a process-passing language and studying termination through encoding into  $\pi$ .
  - ▶ Complexity of inference for weight-based type systems (TGC'07).
    - ▶ "Given  $P$ , how efficiently can we compute whether  $\vdash P$ ?"
  - ▶ Refinements in order to reach greater expressiveness (IFIP/TCS'08)
    - ▶ Static/Dynamic analysis: typing diverging processes, controlling their executions.

# Future works

## A versatile method

- ▶ A **termination proof** as an argument of the method.
- ▶ **Pure functional**  $\pi \rightsquigarrow$  any terminating **functional** calculus
  - ▶ Functional calculus with integer recursions:  
def  $f = (n, x).P_1$  in  $P_2$   
well-typed if  $f$  appears in  $P_1$  only in  $\bar{f}\langle n - 1, v \rangle$ .
  - ▶ Polymorphism (types can be quantified, not levels).

## Complexity for processes

- ▶ Natural question after termination.
- ▶ Defining complexity classes for  $\pi$  processes.
- ▶ Ensuring complexity bounds with type systems.



## Appendix: Typing rules for Deng's S1

$$\text{(Nil)} \frac{}{\vdash \mathbf{0} : 0}$$

$$\text{(Par)} \frac{\vdash P_1 : n_1 \quad \vdash P_2 : n_2}{\vdash P_1 \mid P_2 : \max(n_1, n_2)}$$

$$\text{(Res)} \frac{\vdash P : n \quad \Gamma(a) = \#^k T}{\vdash (\nu a) P : n}$$

$$\text{(Out)} \frac{\vdash P : n \quad a : \#^k T \quad v : T}{\vdash \bar{a}\langle v \rangle.P : \max(n, k)}$$

$$\text{(In)} \frac{\vdash P : n \quad a : \#^k T \quad x : T}{\vdash a(x).P : n}$$

$$\text{(Rep)} \frac{\vdash P : n \quad a : \#^k T \quad x : T \quad k > n}{\vdash !a(x).P : 0}$$

## Appendix: Typing rules for Deng's S3

$$\text{(Nil)} \frac{}{\vdash_{\Gamma}^{\kappa} \mathbf{0} : \emptyset} \qquad \text{(Par)} \frac{\vdash_{\Gamma}^{\kappa} P_1 : M_1 \quad \vdash_{\Gamma}^{\kappa} P_2 : M_2}{\vdash_{\Gamma}^{\kappa} P_1 \mid P_2 : M_1 \uplus M_2}$$

$$\text{(Res)} \frac{\vdash_{\Gamma}^{\kappa} P : M \quad \Gamma(a) = \#^k T}{\vdash_{\Gamma}^{\kappa} (\nu a) P : M}$$

$$\text{(In)} \frac{\vdash_{\Gamma}^{\kappa} P : M \quad \Gamma(a) = \#^k T \quad \Gamma(x) = T}{\vdash_{\Gamma}^{\kappa} a(x).P : M}$$

$$\text{(Out)} \frac{\vdash_{\Gamma}^{\kappa} P : M \quad \Gamma(a) = \#^k T \quad \Gamma(v) = T}{\vdash_{\Gamma}^{\kappa} \bar{a}\langle v \rangle.P : M \uplus \{k\}}$$

$$\text{(Rep)} \frac{\vdash_{\Gamma}^{\kappa} P : M \quad \forall i, \Gamma(a_i) = \#^{k_i} T_i \wedge \Gamma(x_i) = T_i \quad M <_{\text{mul}} \{k_1, \dots, k_n\}}{\vdash_{\Gamma}^{\kappa} !a_1(x_1) \dots a_n(x_n).P : \emptyset}$$

## Appendix: Typing rule for Deng's S4

$$(\mathbf{PoNil}) \frac{}{\Gamma, \mathcal{R} \vdash_{\text{po}} \mathbf{0} : \emptyset} \quad (\mathbf{PoRes}) \frac{\Gamma, \mathcal{R} \vdash_{\text{po}} P : M}{\Gamma, \mathcal{R} - \{c\} \vdash_{\text{po}} (\nu c) P : M}$$

$$(\mathbf{PoPar}) \frac{\Gamma, \mathcal{R} \vdash_{\text{po}} P_1 : M_1 \quad \Gamma, \mathcal{R} \vdash_{\text{po}} P_2 : M_2}{\Gamma, \mathcal{R} \vdash_{\text{po}} P_1 \mid P_2 : M_1 \uplus M_2}$$

$$(\mathbf{PoOut}) \frac{\Gamma, \mathcal{R} \vdash_{\text{po}} P : M \quad \Gamma(a) = \#_{\mathcal{R}_a}^k \tilde{T} \quad \Gamma(\tilde{v}) = T_i \quad \mathcal{R}_a[\tilde{v}] \subseteq \mathcal{R}}{\Gamma, \mathcal{R} \vdash_{\text{po}} \bar{a}(\tilde{v}).P : M \uplus \{a\}}$$

$$(\mathbf{PoIn}) \frac{\Gamma, \mathcal{R}' \vdash_{\text{po}} P : M \quad \Gamma(a) = \#_{\mathcal{R}_a}^k \tilde{T} \quad \Gamma(\tilde{x}) = \tilde{T} \quad \mathcal{R}' = \mathcal{R}_a[\tilde{x}] \uplus \mathcal{R}}{\Gamma, \mathcal{R} \vdash_{\text{po}} a(\tilde{x}).P : M}$$

$$(\mathbf{PoRep}) \frac{\Gamma, \mathcal{R}' \vdash_{\text{po}} P : M \quad \Gamma(a_i) = \#_{\mathcal{R}_{a_i}}^{k_i} \tilde{T}_i \quad \Gamma(\tilde{x}_i) = \tilde{T}_i \quad \mathcal{R}' = \mathcal{R}_{a_1}[\tilde{x}_1] \uplus \dots \uplus \mathcal{R}_{a_n}[\tilde{x}_n] \uplus \mathcal{R} \quad \{a_1, \dots, a_n\} \text{mul}_{(>1v1, \mathcal{R})} M \quad \text{safe}(\{a_1, \dots, a_n\}, M, P)}{\Gamma, \mathcal{R} \vdash_{\text{po}} !a_1(\tilde{x}_1) \dots a_n(\tilde{x}_n).P : \emptyset}$$

## Appendix: NP-completeness reduction

The level associated to name  $T$  is noted  $t$ .

- ▶ The constraint associated to  $!T.T.\overline{x_k}.\overline{x'_k}$  is equivalent to

$$(t \geq \text{Lvl}(x_k) \wedge t \geq \text{Lvl}(x'_k)) \wedge (t > \text{Lvl}(x_k) \vee t > \text{Lvl}(x'_k)) .$$

The constraint associated to  $!x_k.x'_k.T$  is equivalent to

$$t \leq \text{Lvl}(x_k) \vee t \leq \text{Lvl}(x'_k) .$$

Hence, the constraint determined by  $P_k$  is equivalent to

$$(\text{Lvl}(x_k) = t \wedge \text{Lvl}(x'_k) < t) \vee (\text{Lvl}(x'_k) = t \wedge \text{Lvl}(x_k) < t) . \quad (1)$$

- ▶ The constraint associated to  $!n_{i_1}.n_{i_2}.n_{i_3}.\overline{T}$  is equivalent to

$$t \leq \text{Lvl}(n_i^1) \vee t \leq \text{Lvl}(n_i^2) \vee t \leq \text{Lvl}(n_i^3) . \quad (2)$$

## Appendix: Technical details of impure $\pi$ rules

- ▶ Non-replicated imperative inputs treated like the replicated ones:  $\text{def } f^1 = a^1(x).(\overline{x^1} \mid \overline{a^1}\langle x \rangle) \text{ in } (\overline{f^1} \mid \overline{a^1}\langle f \rangle)$
- ▶ Allowing  $k < n$  is dangerous:  $\text{def } f^1 = \overline{a^2} \text{ in } (!a^2.\overline{f^1} \mid \overline{a^2})$ .

## Appendix: impure $\pi$ : outputs and other rules

Types for names:  $T ::= \mathbf{b} \mid \#_F^k T \mid \#_I^k T$

$$\frac{}{\vdash \mathbf{0} : 0} \quad \frac{\vdash P : n}{\vdash (\nu a) P : n} \quad \frac{\vdash P_1 : n_1 \quad \vdash P_2 : n_2}{\vdash P_1 \mid P_2 : \max(n_1, n_2)}$$

$$\frac{\vdash P : n \quad \Gamma(v) = \#_{\bullet}^k T \quad \Gamma(w) = T}{\vdash \bar{v}\langle w \rangle.P : \max(k, n)}$$

- ▶ Rule for output

$v : \#_{\bullet}^k T$ :  $v$  is either functional or imperative ( $v = f$  or  $v = a$ ).

## Appendix: Example for impure $\pi$

$$\begin{aligned} \text{Sys}_1 &\stackrel{\text{def}}{=} (\nu \text{lock}_1, \dots, \text{lock}_k) \\ &\quad \left( \text{lock}_1 \mid \dots \mid \text{lock}_k \mid \right. \\ &\quad \quad \text{def } s = (c, x). \bar{c} \langle \mathbf{enc}[c, x] \rangle \text{ in} \\ &\quad \quad \text{def } c_1 = C_1 \text{ in} \\ &\quad \quad \dots \\ &\quad \quad \text{def } c_{k-1} = C_{k-1} \text{ in} \\ &\quad \quad \text{def } c_k = C_k \text{ in} \\ &\quad \quad \left. (\bar{s} \langle c_1, \text{msg}_1 \rangle \mid \dots \mid \bar{s} \langle c_1, \text{msg}_n \rangle) \right) \end{aligned}$$

with:

$$\begin{aligned} C_i &\stackrel{\text{def}}{=} (y_i). \overline{\text{lock}_i}. (\text{lock}_i \mid \bar{s} \langle c_{i+1}, \mathbf{dec}_i[y_i] \rangle) \\ C_k &\stackrel{\text{def}}{=} (y_k). \overline{\text{lock}_k}. (\text{lock}_k \mid \bar{d} \langle \mathbf{dec}_k[y_k] \rangle) \end{aligned}$$

### Levels

$$\text{lock}_i : \#_1^0 \mathbb{K}, \quad c_i : \#_F^1 \mathbf{b}, \quad s : \#_F^1 (\#_F^1 \mathbf{b} \times \mathbf{b}), \quad \text{msg}_i : \mathbf{b}, \quad d : \#_F^1 \mathbf{b}$$

## Appendix: Simulation Lemma

### Simulation Lemma

- ▶ If  $P \xrightarrow{\gamma_n} P'$  for  $n \leq p$  then  $\text{pr}^p(P) \simeq \text{pr}^p(P')$ .
- ▶ If  $P \xrightarrow{\gamma_n} P'$  for  $n < p$  then  $\text{pr}^p(P) \simeq \text{pr}^p(P')$ .
- ▶ If  $P \xrightarrow{\gamma_p} P'$  then  $\text{pr}^p(P) \xrightarrow{\gamma_p} \text{pr}^p(P')$ .

Holds thanks to conditions  $k > l$  and  $k \geq l$  in the typing rules.

## Appendix: Another example for impure $\pi$

$$\begin{aligned} \text{Sys}_2 &\stackrel{\text{def}}{=} \text{def } s = (n, r, f). \\ &\quad ( \text{if } f = \mathbf{tintin} \text{ then } (\nu h) (\bar{r}\langle h \rangle.\bar{h}) \\ &\quad \quad | \text{if } f = \mathbf{asterix} \text{ then } \dots \\ &\quad \quad \dots \\ &\quad \quad | \text{if } n > 0 \text{ then } \bar{s}\langle n-1, r, f \rangle ) \\ &\text{in } (\nu r') ( \quad \bar{s}\langle 15, r', \mathbf{tintin} \rangle \\ &\quad \quad | (\nu c) (\bar{c} \mid !c.r'(z).(\bar{c} \mid z)) ) \end{aligned}$$

### Levels

$f, \mathbf{tintin}, \mathbf{asterix} : \mathbf{b}$ ,  $h, z : \mathbf{b}$ ;  $r, r' : \#_F^1 \mathbf{b}$ ,  $s : \#_F^2 (\text{nat} \times \#_F^1 \mathbf{b} \times \mathbf{b})$ ,  $c : \#_I^0 \mathbf{b}$

## Appendix: Reductions for $\lambda_{\text{ref}}$

$$(\beta) \frac{}{(\lambda x. M \ V, \delta) \mapsto (M\{V/x\}, \delta)}$$

$$(\text{ref}) \frac{u \notin \text{supp}(\delta) \quad \vdash_{\Gamma} V : (T, -)}{(\text{ref}_n \ V, \delta) \mapsto (u, (\delta \langle u \rightsquigarrow V \rangle))}$$

$$(\text{deref}) \frac{\delta(u) = V}{(\text{deref}_n(u), \delta) \mapsto (V, \delta)}$$

$$(\text{store}) \frac{\vdash_{\Gamma} V : (T, -)}{(u :=_n V, (\delta)) \mapsto (*, (\delta \langle u \rightsquigarrow V \rangle))}$$

$$(\text{context}) \frac{(M, \delta) \mapsto (M', \delta')}{(\mathbf{E}[M], \delta) \mapsto (\mathbf{E}[M'], \delta')}$$

## Appendix: Typability for $\lambda_{\text{ref}}$

$$\text{(App)} \frac{\vdash_{\Gamma} M : (T_1 \rightarrow^n T_2, m) \quad \vdash_{\Gamma} N : (T_1, k)}{\vdash_{\Gamma} M N : (T_2, \max(m, n, k))}$$

$$\text{(Abs)} \frac{\vdash_{\Gamma, x:T_1} M : (T_2, n) \quad \Gamma(x) = T_1}{\vdash_{\Gamma} \lambda x. M : (T_1 \rightarrow^n T_2, 0)}$$

$$\text{(Ref)} \frac{\vdash_{\Gamma} M : (T_1, m)}{\vdash_{\Gamma} \text{ref}_n M : (T_1 \text{ ref}_n, \max(n, m))}$$

$$\text{(Var)} \frac{\Gamma(x) = T_1}{\vdash_{\Gamma} x : (T_1, 0)}$$

$$\text{(Uni)} \frac{}{\vdash_{\Gamma} \star : (\mathbb{K}, 0)}$$

$$\text{(Add)} \frac{}{\vdash_{\Gamma} u_{(n, T_1)} : (T_1 \text{ ref}_n, 0)}$$

$$\text{(Asg)} \frac{\vdash_{\Gamma} M : (T_1 \text{ ref}_n, m) \quad \vdash_{\Gamma} N : (T_1, k)}{\vdash_{\Gamma} M :=_n N : (\mathbb{K}, \max(m, n, k))}$$

$$\text{(Drf)} \frac{\vdash_{\Gamma} M : (T \text{ ref}_n, m)}{\vdash_{\Gamma} \text{deref}_n(M) : (T, \max(m, n))}$$

$$\text{(Emp)} \frac{}{\vdash_{\Gamma} \emptyset}$$

$$\text{(Sto)} \frac{\vdash_{\Gamma} \delta \quad \vdash_{\Gamma} V : (T, 0)}{\vdash_{\Gamma} \delta \langle u_{(n, T)} \rightsquigarrow V \rangle}$$

## Appendix: Pruning for $\lambda_{\text{ref}}$

If  $M$  is not related to  $p$ :

$$\text{pr}^P(M) = \mathbf{v}_T$$

Otherwise:

$$\text{pr}^P(M_1 M_2) = \text{pr}^P(M_1) \text{pr}^P(M_2)$$

$$\text{pr}^P(x) = x$$

$$\text{pr}^P(\lambda x. M_1) = \lambda x. \text{pr}^P(M_1)$$

$$\text{pr}^P(\text{ref}_n M_1) = (\Pi^{(1,2)} \star \text{pr}^P(M_1))$$

$$\text{pr}^P(\text{deref}_n(M_1)) = (\Pi^{(1,2)} \mathbf{v}_T \text{pr}^P(M_1))$$

$$\text{pr}^P(M_1 :=_n M_2) = (\Pi^{(1,3)} \star \text{pr}^P(M_1) \text{pr}^P(M_2))$$

$$\text{pr}^P(u_{(n, T_1)}) = \star$$

## Appendix: 2 Lemmas for $\lambda_{\text{ref}}$

### Dividing evaluation contexts

Let  $\mathbf{E}$  be an evaluation context and  $p$  an integer.

1. Either for all  $M$ ,  $\text{pr}^p(\mathbf{E}[M]) = \text{pr}^p(\mathbf{E})[\text{pr}^p(M)]$
2. Or there exists  $\mathbf{E}_1$  and  $\mathbf{E}_2 \neq []$  s.t.  $\mathbf{E} = \mathbf{E}_1[\mathbf{E}_2]$  and, for all  $M$ :
  - 2.1 If  $M$  has effect  $\geq p$ , then
$$\text{pr}^p(\mathbf{E}[M]) = \text{pr}^p(\mathbf{E}[M]) = \text{pr}^p(\mathbf{E})[\text{pr}^p(M)].$$
  - 2.2 If  $M$  has effect  $< p$ , then  $\text{pr}^p(\mathbf{E}[M]) = \text{pr}^p(\mathbf{E}_1)[V_{T''}]$   
(where  $T''$  is the type of  $\mathbf{E}_2$ ).

### Context reduction

If  $\vdash_{\Gamma} \mathbf{E}_2 : (T'', m)$  and  $\mathbf{E}_2$  is not related with level  $p$ , for all terms  $M, M'$ ,

1.  $\text{pr}^p(\mathbf{E}_2)[(\Pi^{(1,2)} V_T M)] \rightarrow^+ V_{T''}$ ;
2.  $\text{pr}^p(\mathbf{E}_2)[(\Pi^{(1,3)} V_T M M')] \rightarrow^+ V_{T''}$ .

# Appendix: Implicit complexity for processes

with Ugo Dal Lago (University of Bologna, Italy).

## Principles

- ▶ **Implicit complexity**: syntactical constraint allowing only programs executing under a complexity bounds to e defined.
- ▶ Defining a notion of complexity class for processes.
  - ▶  $P \in \mathbf{Class}(f)$  when  $P \xrightarrow{a(\tilde{x})} P'$ ,  $P'$  makes at most  $f(|\tilde{x}|)$  reductions.
- ▶ Taking causality into accounts (multiple requests, shared ressource, locks)
- ▶ Defining a type system s.t.  $\vdash P$  implies  $P \in \mathbf{Class}(f)$  for some elementary/polynomial function  $f$ .
  - ▶ Completeness ?

# Appendix: Information flow for higher-order concurrent system

with Catuscia Palamidessi (LiX, Palaiseau)

## Principles

- ▶ Considering processes as *Information channel*
- ▶ Degree of protection: how easily secret inputs  $s$  can be deduced from observable actions  $o_1, o_2$  ?:

$$s_1 \cdot (\overline{o_1} \oplus_{1/2} \overline{o_2}) + s_2 \cdot (\overline{o_1} \oplus_{1/3} \overline{o_2})$$

- ▶ Is | secure ?

$$\Pr(P_1 | P_2) = \min_{i \in \{1,2\}} (\Pr(P_i))$$

- ▶ Existing results in  $\text{CCS}^P \rightsquigarrow$  results  $\text{HO}\pi_2^P$

# Appendix: Subtyping for $\lambda$ -encoding

with Kohei Honda (Queen Mary, University of London)

## Principles

- ▶  $\pi^{\text{aff}}$ , an affine  $\pi$  calculus
- ▶ Seeing functional computation as interaction:

$$\llbracket \alpha \longrightarrow \beta \rrbracket = \&_{i \in I}(\tilde{T}_i, \overline{\llbracket \beta \rrbracket}) \text{ if } \llbracket \alpha \rrbracket = \oplus_{i \in I}(\tilde{T}_i)$$

for instance:

$$\llbracket \text{Bool} \longrightarrow \beta \rrbracket = \text{true} \overline{\llbracket \beta \rrbracket} \& \text{false} \overline{\llbracket \beta \rrbracket}$$

- ▶ Encoding typed  $\lambda$  into  $\pi^{\text{aff}}$ .
- ▶ Studying the notion of subtyping through the encoding.