

# A logical view on scheduling in concurrency

Emmanuel Beffara

I2M, CNRS & Université d'Aix-Marseille

Chocola – 3 avril 2014

## Introduction

- Proofs as processes
- Processes as untyped proofs
- Why we should search further

## Proofs as schedules

- MLL with actions
- Soundness and completeness

## Uniform translations

- Asynchronous translation
- Synchronous translation

## Discussion

# Logic vs computation

- The *formulae as types* approach:

formula  $\leftrightarrow$  type  
proof rules  $\leftrightarrow$  primitive instructions  
proof  $\leftrightarrow$  program  
normalization  $\leftrightarrow$  evaluation

- The *proof search* approach:

formula  $\leftrightarrow$  program  
proof rules  $\leftrightarrow$  operational semantics  
proof construction  $\leftrightarrow$  execution  
proof  $\leftrightarrow$  successful run

# (Logic vs computation) vs concurrency

- The *formulae as types* approach:

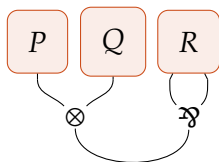
formula  $\leftrightarrow$  type  
proof rules  $\leftrightarrow$  primitive instructions  
proof  $\leftrightarrow$  program  
normalization  $\leftrightarrow$  evaluation

- The *proof search* approach:

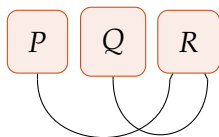
formula  $\leftrightarrow$  program  
proof rules  $\leftrightarrow$  operational semantics  
proof construction  $\leftrightarrow$  execution  
proof  $\leftrightarrow$  successful run

- How can we fit *concurrency* into this framework?  
What is a proper *denotational semantics* for concurrency?

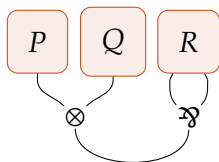
- Cut elimination in proof nets is an interactive process:



- Cut elimination in proof nets is an interactive process:



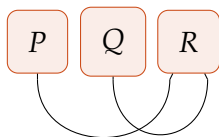
- Cut elimination in proof nets is an interactive process:



- It is natural to represent it a language for interactive processes:

$$(vz)\left((vxy)(\bar{z}\langle xy \rangle \mid P \mid Q) \mid z(xy)R\right)$$

- Cut elimination in proof nets is an interactive process:



- It is natural to represent it a language for interactive processes:

$$(vz)\left((vxy)(\bar{z}\langle xy \rangle \mid P \mid Q) \mid z(xy)R\right) \rightarrow (vxy)(P \mid Q \mid R)$$



This idea was first implemented in



Gianluigi Bellin and Phil Scott

On the  $\pi$ -calculus and linear logic

Theoretical Computer Science, 1994

This idea was first implemented in



Gianluigi Bellin and Phil Scott

On the  $\pi$ -calculus and linear logic

Theoretical Computer Science, 1994

Good points:

- Adequate representation of proof dynamics
- Study of information flow through proofs

# Proofs as processes

This idea was first implemented in



Gianluigi Bellin and Phil Scott

On the  $\pi$ -calculus and linear logic

Theoretical Computer Science, 1994

Good points:

- Adequate representation of proof dynamics
- Study of information flow through proofs

Limitations:

- Requires a lot of coding
- Touches processes of a very restricted form
- Does not provide much insight on the  $\pi$ -calculus

# Typing processes in linear logic

Axiom and cut:

$$\frac{}{u \rightarrow v \vdash u : \downarrow A^\perp, v : \uparrow A} \quad \frac{P \vdash \Gamma, \vec{x} : A \quad Q \vdash \vec{x} : A^\perp, \Delta}{(v\vec{x})(P \mid Q) \vdash \Gamma, \Delta}$$

Multiplicatives:

$$\frac{P \vdash \Gamma, \vec{x} : A \quad Q \vdash \vec{y} : B, \Delta}{P \mid Q \vdash \Gamma, \vec{x}\vec{y} : A \otimes B, \Delta} \quad \frac{P \vdash \Gamma, \vec{x} : A, \vec{y} : B}{P \vdash \Gamma, \vec{x}\vec{y} : A \wp B}$$

Actions:

$$\frac{P \vdash \Gamma, \vec{x} : A}{u(\vec{x}).P \vdash \Gamma, u : \downarrow A} \quad \frac{P \vdash \Gamma, \vec{x} : A}{\bar{u}(\vec{x}).P \vdash \Gamma, u : \uparrow A}$$

Exponentials for replication, additives for external choice.

# Typing processes in linear logic

The system on the previous slide was introduced in



EB

A concurrent model for linear logic

MFPS 2006

but was found to be strongly related to



Nobuko Yoshida, Martin Berger, and Kohei Honda

Strong normalisation in the  $\pi$ -calculus

LICS 2001

Bellin and Scott's encoding decomposes inside.

Independently developped:



Luís Caires and Frank Pfenning

Session types as intuitionistic linear propositions

Concur 2010

appears as a fragment.

# Typing processes in linear logic

Good things:

- Typed processes cannot diverge or deadlock.
- Typing is preserved by reduction  
*up to structural congruence.*
- Extends to differential linear logic  
*through “algebraic” extensions of process calculi.*
- Induces translations of the  $\lambda$ -calculus into the  $\pi$ -calculus.

# Typing processes in linear logic

## Good things:

- Typed processes cannot diverge or deadlock.
- Typing is preserved by reduction  
*up to structural congruence.*
- Extends to differential linear logic  
*through “algebraic” extensions of process calculi.*
- Induces translations of the  $\lambda$ -calculus into the  $\pi$ -calculus.

## Shortcomings:

- Typed processes are essentially functional.
- Only *top-level* cut elimination matches execution.
- Many well-behaved interaction patterns are not typable.

$$a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$$

# Processes as proofs

Dual approach: implement processes as proofs in a suitable logic.



# Processes as untyped proofs

Dual approach: implement processes as proofs in a suitable logic.

- Translating *all* processes requires an untyped proof language.

# Processes as untyped proofs

Dual approach: implement processes as proofs in a suitable logic.

- Translating *all* processes requires an untyped proof language.
- Standard linear logic is not an option because of confluence.


# Processes as untyped proofs

Dual approach: implement processes as proofs in a suitable logic.

- Translating *all* processes requires an untyped proof language.
- Standard linear logic is not an option because of confluence.
- Differential linear logic allows for explicit non-determinism:

$$\frac{P \vdash \Gamma \quad Q \vdash \Gamma}{P + Q \vdash \Gamma}$$

Its rules allow for an implementation of all processes.

-  **Thomas Ehrhard and Olivier Laurent**  
Interpreting a finitary  $\pi$ -calculus in differential interaction nets  
Concur 2007

# Processes as untyped proofs

Good points:

- Does provide insights on concurrent processes
- Relates algebraic proof semantics and process semantics

# Processes as untyped proofs

Good points:

- Does provide insights on concurrent processes
- Relates algebraic proof semantics and process semantics

Limitations:

- Not clear how to get logic back into the process language
- Prefixing is only described very indirectly:

$\pi$ -calculus  $\longrightarrow$  solos calculus  $\longrightarrow$  differential nets

# A few observations

Proof normalization, aka *cut elimination*:

- the meaning of a proof is in its normal form,
- normalization is an *explicitation* procedure,
- it really wants to be confluent.

Interpretation of concurrent processes:

- the meaning is the *interaction*, the final (irreducible) state is less relevant,
- a given process may behave very differently depending on scheduling decisions.

*Some information is missing.*

The principles of our new interpretation:

formula  $\leftrightarrow$  type of interaction  
proof rules  $\leftrightarrow$  primitives for building schedules  
proof  $\leftrightarrow$  schedule for a program  
normalization  $\leftrightarrow$  evaluation according to a schedule

This is not exactly:

- *Curry-Howard* for processes:

proofs are not programs, but behaviours of programs

- *Proof search*:

the dynamics is not in proof construction but in cut-elimination

but a sort of middle ground in between.

# Proofs as schedules: step 1

The first step: a logical description of all executions.



EB and Virgile Mogbil

Proofs as executions

IFIP TCS 2012 — Chocla 14/3/2013

How we proceed:

- Back to CCS, for now.
- Slightly change the logic to represent actions explicitly.
- Match each execution with cut elimination of some proof.



# Multiplicative CCS

We consider a CCS-style process calculus.

$P, Q := 1$	inaction
$a.P$	perform $a$ then do $P$
$P \mid Q$	interaction of $P$ and $Q$
$(\nu a)P$	scope restriction

There is one source of non-determinism:  
the pairing of associated events upon synchronization

$$a.P \mid a.Q \mid \bar{a}.R \rightarrow \begin{cases} a.P \mid Q \mid R \\ P \mid a.Q \mid R \end{cases}$$

# MLL with actions

## The formulas

Types of schedules:

$A, B := \langle a \rangle A$	do action $a$ and then act as $A$
$A \otimes B$	two independent parts, one as $A$ , the other as $B$
$A \wp B$	$A$ and $B$ are both exhibited, but correlated
$\alpha$	an unspecified behaviour (type variable)
$\alpha^\perp$	something that can interact with $\alpha$
$(\forall \alpha A, \exists \alpha A)$	quantification over behaviours )

Transforming schedules:

$A_1, \dots, A_n \vdash B$	behave as type $B$ in association with processes behaving as each type $A_i$
----------------------------	---

Two-sided version.

# MLL with actions

## The formulas

Types of schedules:

$A, B := \langle a \rangle A$	do action $a$ and then act as $A$
$A \otimes B$	two independent parts, one as $A$ , the other as $B$
$A \wp B$	$A$ and $B$ are both exhibited, but correlated
$\alpha$	an unspecified behaviour (type variable)
$\alpha^\perp$	something that can interact with $\alpha$
$(\forall \alpha A, \exists \alpha A)$	quantification over behaviours)

Transforming schedules:

$\vdash A_1^\perp, \dots, A_n^\perp, B$	behave as type $B$ in association with processes behaving as each type $A_i$
---	---

Duality:  $(A \otimes B)^\perp = A^\perp \wp B^\perp$ ,  $(\langle a \rangle A)^\perp = \langle \bar{a} \rangle (A^\perp)$ .

# MLL with actions

## The formulas

Types of schedules:

$A, B := \langle a \rangle A$	do action $a$ and then act as $A$
$A \otimes B$	two independent parts, one as $A$ , the other as $B$
$A \wp B$	$A$ and $B$ are both exhibited, but correlated
$\alpha$	an unspecified behaviour (type variable)
$\alpha^\perp$	something that can interact with $\alpha$
$(\forall \alpha A, \exists \alpha A)$	quantification over behaviours )

Transforming schedules:

$P \vdash A_1^\perp, \dots, A_n^\perp, B$       $P$  *can* behave as type  $B$  in association  
with processes behaving as each type  $A_i$

Duality:  $(A \otimes B)^\perp = A^\perp \wp B^\perp$ ,  $(\langle a \rangle A)^\perp = \langle \bar{a} \rangle (A^\perp)$ .

# MLL with actions

## Proof rules

Axiom and cut:

$$\frac{}{1 \vdash A^\perp, A} \quad \frac{P \vdash \Gamma, A \quad Q \vdash A^\perp, \Delta}{P|Q \vdash \Gamma, \Delta}$$

Multiplicatives:

$$\frac{P \vdash \Gamma, A \quad Q \vdash B, \Delta}{P|Q \vdash \Gamma, A \otimes B, \Delta} \quad \frac{P \vdash \Gamma, A, B}{P \vdash \Gamma, A \wp B}$$

Actions:

$$\frac{P \vdash \Gamma, A}{a.P \vdash \Gamma, \langle a \rangle A}$$

Quantification:

$$\frac{P \vdash \Gamma, A \quad \alpha \notin \text{fv}(\Gamma)}{P \vdash \Gamma, \forall \alpha A} \quad \frac{P \vdash \Gamma, A[B/\alpha]}{P \vdash \Gamma, \exists \alpha A}$$

Ceci n'est pas un système de types.

# MLL with actions

## Proof rules

Axiom and cut:

$$\frac{}{1 \vdash A^\perp, A} \quad \frac{P \vdash \Gamma, A \quad Q \vdash A^\perp, \Delta}{P | Q \vdash \Gamma, \Delta}$$

Multiplicatives:

$$\frac{P \vdash \Gamma, A \quad Q \vdash B, \Delta}{P | Q \vdash \Gamma, A \otimes B, \Delta} \quad \frac{P \vdash \Gamma, A, B}{P \vdash \Gamma, A \wp B}$$

Actions:

$$\frac{P \vdash \Gamma, A}{a.P \vdash \Gamma, \langle a \rangle A}$$

Quantification:

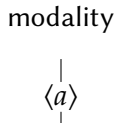
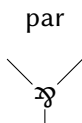
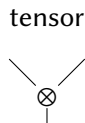
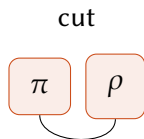
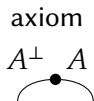
$$\frac{P \vdash \Gamma, A \quad \alpha \notin \text{fv}(\Gamma)}{P \vdash \Gamma, \forall \alpha A} \quad \frac{P \vdash \Gamma, A[B/\alpha]}{P \vdash \Gamma, \exists \alpha A}$$

Ceci n'est pas un système de types.

# MLL with actions

## Proof nets

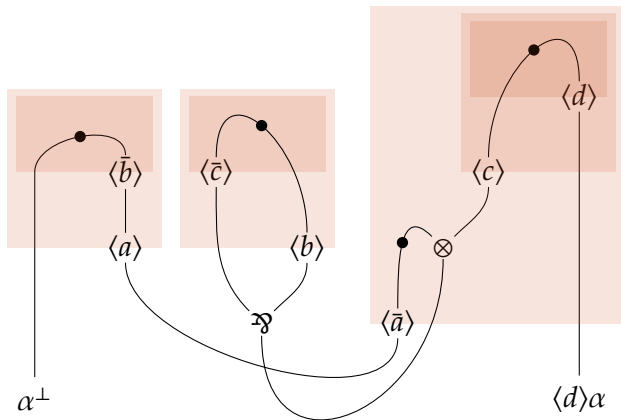
MLLa admits proof nets: those of MLL plus unary links for modalities.



- Modality rules commute with everything, indeed  $A \simeq \langle a \rangle A$ .
- Correctness criteria: the same as MLL.
- We avoid second-order quantification for simplicity, we stick with parametricity in type variables.

# The cyclic example

The following proof is an annotation for  $a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$ :



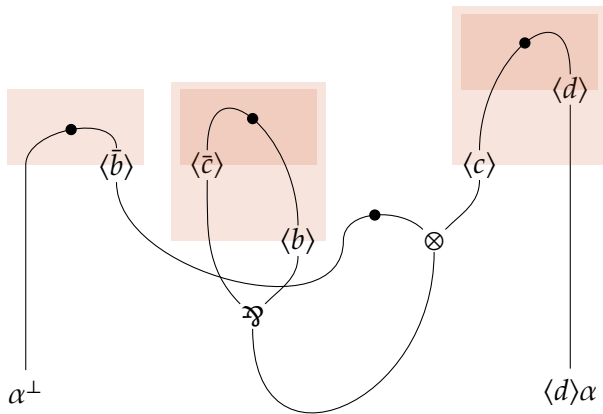
If we use boxes, we have a “head cut elimination” matching execution:

$$a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$$



# The cyclic example

The following proof is an annotation for  $a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$ :

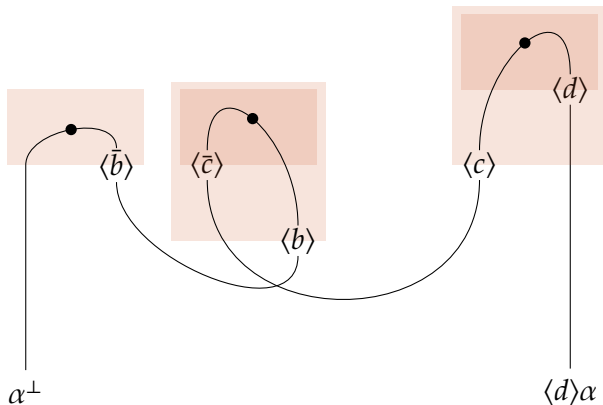


If we use boxes, we have a “head cut elimination” matching execution:

$$a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d \rightarrow \bar{b} \mid b.\bar{c} \mid c.d$$

# The cyclic example

The following proof is an annotation for  $a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$ :

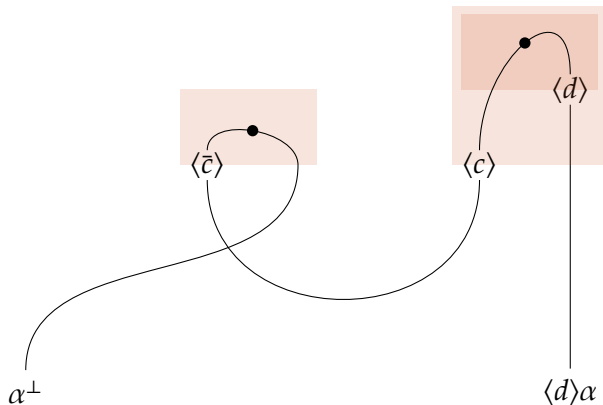


If we use boxes, we have a “head cut elimination” matching execution:

$$a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d \rightarrow \bar{b} \mid b.\bar{c} \mid c.d$$

# The cyclic example

The following proof is an annotation for  $a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$ :

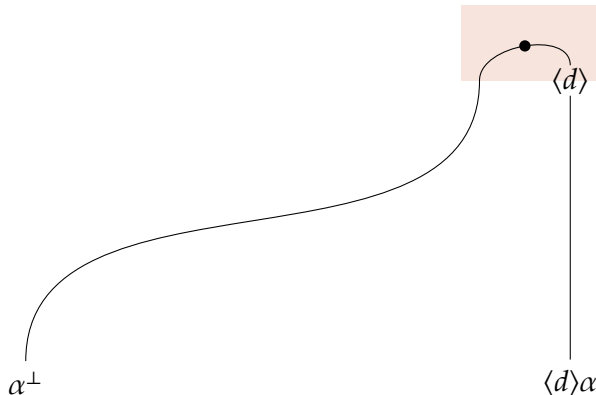


If we use boxes, we have a “head cut elimination” matching execution:

$$a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d \rightarrow \bar{b} \mid b.\bar{c} \mid c.d \rightarrow \bar{c} \mid c.d$$

# The cyclic example

The following proof is an annotation for  $a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d$ :



If we use boxes, we have a “head cut elimination” matching execution:

$$a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c.d \rightarrow \bar{b} \mid b.\bar{c} \mid c.d \rightarrow \bar{c} \mid c.d \rightarrow d$$

# The results of step 1

## Theorem (Soundness)

*Typing is preserved by reduction,  
head cut-elimination steps correspond to execution steps.*

The definition of “head” cut-elimination requires boxes for modality rules, to keep track of prefixing.

## Theorem (Completeness)

*For every lock-avoiding run  $P_1 \rightarrow \dots \rightarrow P_n$  there are annotations such that  $\pi_1 : P_1 \vdash \Gamma \rightarrow \dots \rightarrow \pi_n : P_n \vdash \Gamma$  is a cut elimination sequence.*

# Observations

Every execution correspond to some proof:

- the proof provides a schedule (pairing between actions),
- cut elimination provides actual execution.

These proofs have very different types:

- the type is deduced from the execution, it describes control flow according a particular schedule;
- the type describes a way for a process interacts with its environment,
- no *most general* type.

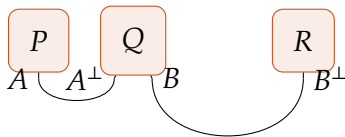
Step 2: make things more uniform.

# The trick for actions prefixes

For annotating a process  $a.P \mid Q \mid \bar{a}.R$  in an execution step

$$a.P \mid Q \mid \bar{a}.R \rightarrow P \mid Q \mid R$$

one may need some plumbing:

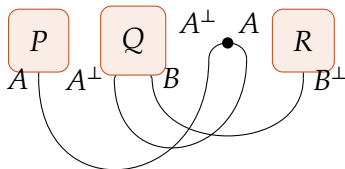


# The trick for actions prefixes

For annotating a process  $a.P \mid Q \mid \bar{a}.R$  in an execution step

$$a.P \mid Q \mid \bar{a}.R \rightarrow P \mid Q \mid R$$

one may need some plumbing:



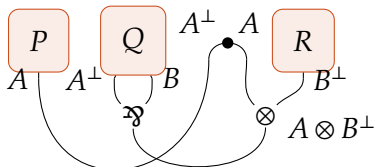


# The trick for actions prefixes

For annotating a process  $a.P \mid Q \mid \bar{a}.R$  in an execution step

$$a.P \mid Q \mid \bar{a}.R \rightarrow P \mid Q \mid R$$

one may need some plumbing:

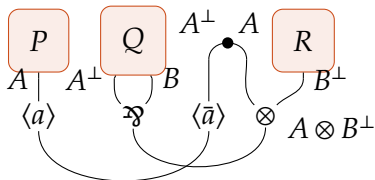


# The trick for actions prefixes

For annotating a process  $a.P \mid Q \mid \bar{a}.R$  in an execution step

$$a.P \mid Q \mid \bar{a}.R \rightarrow P \mid Q \mid R$$

one may need some plumbing:

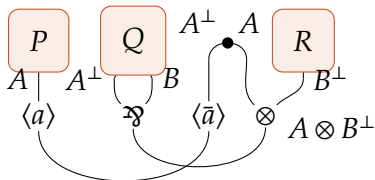


# The trick for actions prefixes

For annotating a process  $a.P \mid Q \mid \bar{a}.R$  in an execution step

$$a.P \mid Q \mid \bar{a}.R \rightarrow P \mid Q \mid R$$

one may need some plumbing:



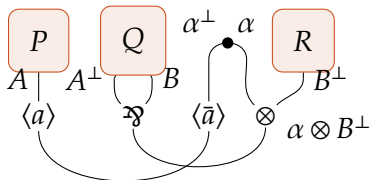
The type of  $\bar{a}.R$  depends on that of  $Q$ ,  
even if only  $Q$  only interacts with  $P$ .

# The trick for actions prefixes

For annotating a process  $a.P \mid Q \mid \bar{a}.R$  in an execution step

$$a.P \mid Q \mid \bar{a}.R \rightarrow P \mid Q \mid R$$

one may need some plumbing:



The construction does not depend on the types: *parametricity in  $\alpha$*   
one can always proceed the same way.

# Type assignment

“Asynchronous” version

## Definition

Terms of M CCS are translated into MLLa formulas as follows:

$$\llbracket 1 \rrbracket_a := \forall \alpha \alpha^\perp \wp \alpha$$

$$\llbracket P \mid Q \rrbracket_a := \llbracket P \rrbracket_a \otimes \llbracket Q \rrbracket_a$$

$$\llbracket a.P \rrbracket_a := \forall \alpha \langle a \rangle \alpha^\perp \wp (\llbracket P \rrbracket_a \otimes \alpha) \quad = \forall \alpha \langle \bar{a} \rangle \alpha \multimap (\llbracket P \rrbracket_a \otimes \alpha)$$

$$\llbracket \bar{a}.P \rrbracket_a := \forall \beta (\llbracket P \rrbracket_a \otimes \beta^\perp) \wp \langle \bar{a} \rangle \beta \quad = \forall \beta (\llbracket P \rrbracket_a \multimap \beta) \multimap \langle a \rangle \beta$$

*Name hiding is left aside for now.*

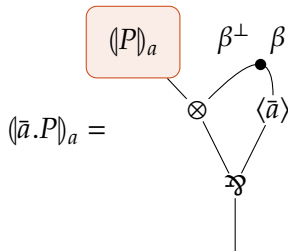
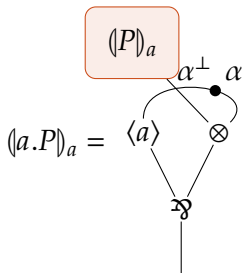
# Proof assignment

“Asynchronous” version

## Fact

For every  $P$ , the type  $[P]_a$  has one cut-free proof  $\langle P \rangle_a$ .

For actions:



# Soundness and completeness

“Asynchronous” version

## Theorem

*There is an execution  $P \rightarrow^* 1$  if and only if  $\llbracket P \rrbracket_a \multimap \llbracket 1 \rrbracket_a$  is provable in MLL (without modality rules).*

# Soundness and completeness

“Asynchronous” version

## Theorem

*There is an execution  $P \rightarrow^* 1$  if and only if  $\llbracket P \rrbracket_a \multimap \llbracket 1 \rrbracket_a$  is provable in MLL (without modality rules).*

From execution to implication:

- each execution step is provable.

From implication to execution:

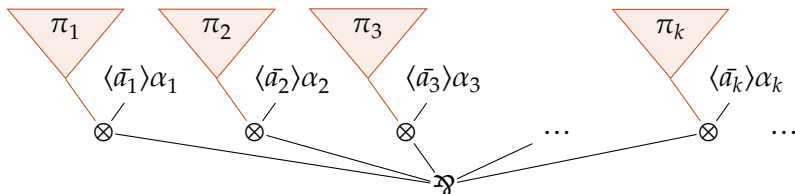
- find a first interaction, exploiting the correctness criterion for a proof of  $\llbracket P \rrbracket_a \multimap \llbracket 1 \rrbracket_a$ .



# Soundness and completeness

“Asynchronous” version: finding the first action

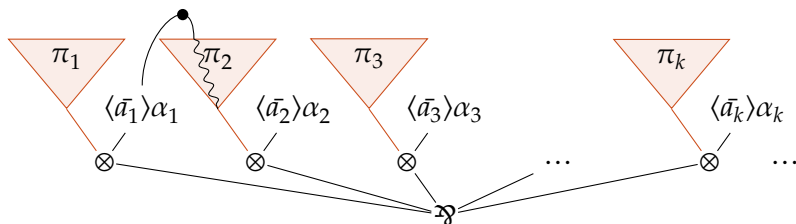
Suppose there is some proof of  $[a_1.P_1 \mid \dots \mid a_n.P_n]_a \multimap [1]_a$  but no two  $a_i$  can synchronize:



# Soundness and completeness

“Asynchronous” version: finding the first action

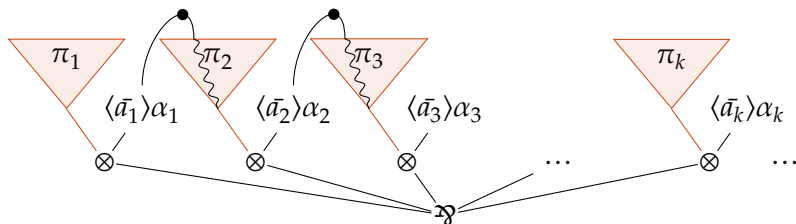
Suppose there is some proof of  $[a_1.P_1 \mid \dots \mid a_n.P_n]_a \multimap [1]_a$  but no two  $a_i$  can synchronize:



# Soundness and completeness

“Asynchronous” version: finding the first action

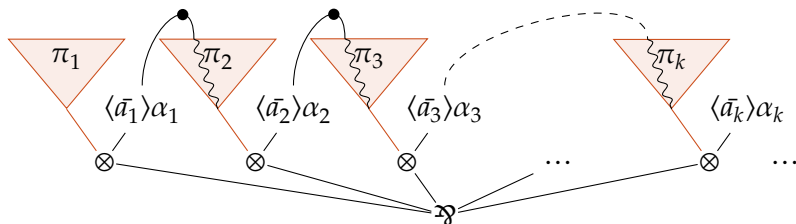
Suppose there is some proof of  $[a_1.P_1 \mid \dots \mid a_n.P_n]_a \multimap [1]_a$  but no two  $a_i$  can synchronize:



# Soundness and completeness

“Asynchronous” version: finding the first action

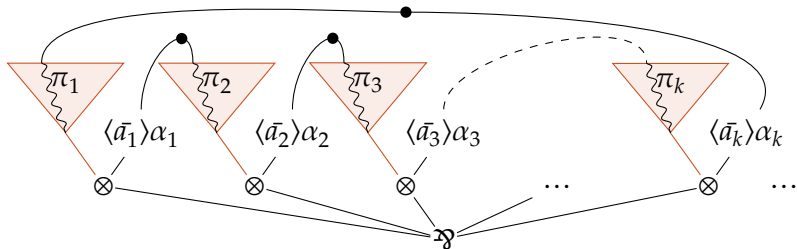
Suppose there is some proof of  $[a_1.P_1 \mid \dots \mid a_n.P_n]_a \multimap [1]_a$  but no two  $a_i$  can synchronize:



# Soundness and completeness

“Asynchronous” version: finding the first action

Suppose there is some proof of  $[a_1.P_1 \mid \dots \mid a_n.P_n]_a \multimap [1]_a$  but no two  $a_i$  can synchronize:



Impossible because of acyclicity!

# Type assignment

“Synchronous” version

## Definition

Terms of M CCS are translated into MLLa formulas as follows:

$$[1]_s := \forall \alpha \alpha^\perp \wp \alpha \qquad = \forall \alpha \alpha \multimap \alpha$$

$$[P \mid Q]_s := [P]_s \otimes [Q]_s$$

$$[a.P]_s := \forall \alpha \langle a \rangle (\alpha^\perp \wp ([P]_s \otimes \alpha)) \qquad = \forall \alpha \langle a \rangle (\alpha \multimap ([P]_s \otimes \alpha))$$

$$[\bar{a}.P]_s := \forall \beta \langle \bar{a} \rangle ([P]_s \otimes \beta^\perp) \wp \beta \qquad = \forall \beta \langle \bar{a} \rangle ([P]_s \multimap \beta) \multimap \beta$$

*Spot the difference!*

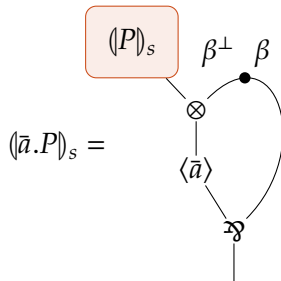
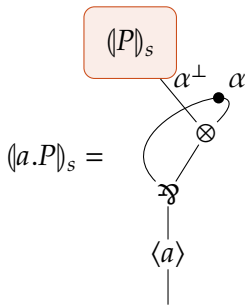
# Proof assignment

“Synchronous” version

## Fact

For every  $P$ , the type  $\llbracket P \rrbracket_s$  has one cut-free proof  $\langle P \rangle_s$ .

For actions:



# Soundness and completeness

“Synchronous” version

## Theorem

*There is an execution  $P \rightarrow^* Q$  if and only if  $\lceil P \rceil_s \dashv\vdash \lceil Q \rceil_s$  is provable in MLL (without modality rules).*



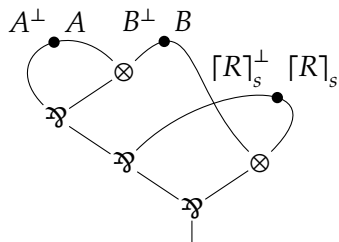
# Soundness and completeness

“Synchronous” version

## Theorem

There is an execution  $P \rightarrow^* Q$  if and only if  $\lceil P \rceil_s \multimap \lceil Q \rceil_s$  is provable in MLL (without modality rules).

From execution to implication:



$$\text{with } \begin{cases} A = \langle a \rangle (\lceil Q \rceil_s^\perp \wp (\lceil P \rceil_s \otimes \lceil Q \rceil_s)) \\ B = \lceil P \rceil_s \otimes \lceil Q \rceil_s \end{cases}$$

proves  $\lceil (a.P \mid \bar{a}.Q) \mid R \rceil_s \multimap \lceil (P \mid Q) \mid R \rceil_s$

# Soundness and completeness

“Synchronous” version

## Theorem

*There is an execution  $P \rightarrow^* Q$  if and only if  $\lceil P \rceil_s \multimap \lceil Q \rceil_s$  is provable in MLL (without modality rules).*

From execution to implication:

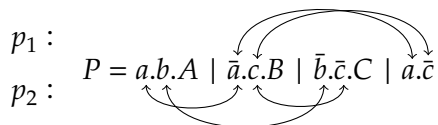
- each execution step is provable.

From implication to execution:

- take a proof of  $\lceil P \rceil_s \multimap \lceil Q \rceil_s$
- cut it against  $\langle P \rangle_s$ , eliminate the cut
- read back process terms from intermediate steps

## Definition

A *pairing* is an association between occurrences of dual actions



## Definition

A *determinisation* of  $P$  along a pairing  $p$  is a renaming  $\partial_p(P)$  of actions in  $P$  where names are equal only for related actions.

$$\partial_{p_1}(P) = a_1.b_1.\partial(A) \mid \bar{a}_2.c_1.\partial(B) \mid \bar{b}_2.\bar{c}_2.\partial(C) \mid a_2.\bar{c}_1$$

$$\partial_{p_2}(P) = a_1.b_1.\partial(A) \mid \bar{a}_1.c_1.\partial(B) \mid \bar{b}_1.\bar{c}_1.\partial(C) \mid a_2.\bar{c}_2$$

# Pairings vs proofs

Facts about pairings:

- each run induces a pairing
- runs are equivalent up to permutation of independent events iff they induce the same pairing
- if  $p$  is a *consistent* pairing of  $P$  then  $p$  is the unique maximal consistent pairing of  $\partial_p(P)$

Hence pairings are *execution schedules* and determinized terms represent them inside the process language.

## Observation

Pairings are related to placements of axiom links in proofs of  $[P]_a \multimap [1]_a$ .

Some points deserve more investigation:

**Replication:** everything extends smoothly by setting  $[!P]_a = ![P]_a$ .

**Choice:** additives are the natural option

**Name hiding:** the situation is not obvious

- use quantifiers?  
existential? nabla?
- partial scheduling?  
 $(\nu a)P$  is  $P$  with some proof that decides what happens on  $a$

**Name passing:** need to fix hiding first!

# Further directions

Current state of affairs:

- A logical description of scheduling in processes
- Explication of *control flow* through processes
- Hints for a new study of prefixing in processes

# Further directions

## Current state of affairs:

- A logical description of scheduling in processes
- Explication of *control flow* through processes
- Hints for a new study of prefixing in processes

## Ongoing questions:

- Which semantics for the logic of schedules?  
*coherence spaces for MLLa, etc*
- CPS-like interpretation of processes?  
*the translation of actions is a kind of double negation*
- A logical account on  $\pi$ -to-solos encoding?  
*by relating to other systems*

Work in progress...