Logical approaches in the semantics of concurrent languages

Emmanuel Beffara

I2M, Université d'Aix-Marseille
LIP, ENS Lyon

LIF, 28th April 2016

Mobile processes and logic

- Process algebras
- Proofs as programs
- Proofs as processes?

Logical description of processes

- A logic for process interaction
- Proofs as schedules
- Typing and safety

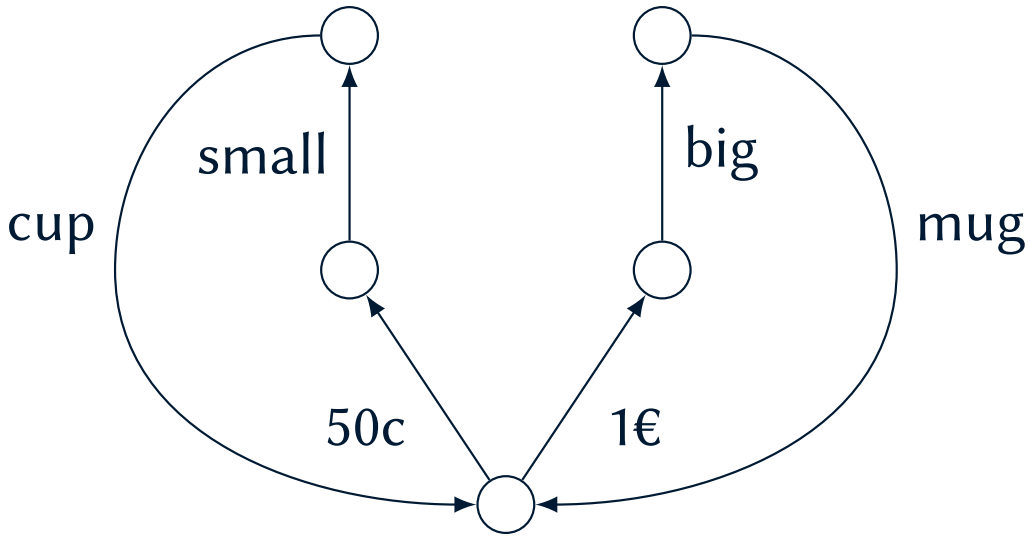
Semantic consequences

1

Mobile processes and logic

Let us start with a coffee

The classical coffee machine example:



the machine: $M = 50c \,.\, small \,.\, \overline{cup} \,.\, M + 1€ \,.\, big \,.\, \overline{mug} \,.\, M$

the client: $C = \overline{50c} \,.\, \overline{small} \,.\, cup$

Process algebra: CCS

We use a formal language to represent programs that interact by message passing:

$$
\begin{array}{rll}
P, Q := & a.P,\ \ \bar{a}.P & \text{action prefix (receive, send)} \\
& P \mid Q,\ \ P + Q,\ \ 0 & \text{parallel composition, choice, stop} \\
& *P & \text{replicated process} \\
& (\nu a)P & \text{name restriction}
\end{array}
$$

One can define the dynamics as a transition system:

$$
\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{\ell} P'}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \frac{P \xrightarrow{\ell} P'}{P + Q \xrightarrow{\ell} P'} \qquad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}
$$

CCS, variations

An alternative way to define the dynamics:

- Set a *structural congruence*:

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid Q \equiv Q \mid P \qquad P \mid 0 \equiv P$$
$$(P + Q) + R \equiv P + (Q + R) \qquad P + Q \equiv Q + P \qquad P + 0 \equiv P$$
$$!P \equiv P \mid !P \qquad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \qquad P \mid (\nu x)Q \equiv (\nu x)(P \mid Q)$$

- Define *reduction* up to structural congruence:

$$(a.P + P') \mid (\bar{a}.Q + Q') \rightarrow P \mid Q$$

One can extend the language with *value passing*:

$$a(x).P + P' \mid \bar{a}\langle v\rangle.Q + Q' \rightarrow P[v/x] \mid Q$$

then the language is extended with elements for computing with values.

Name passing: the π-calculus

Take the variant with value passing:

$$a(x).P \mid \bar{a}\langle v\rangle.Q \rightarrow P[v/x] \mid Q$$

and say that x and v are channel names:

$$a(x).\bar{x}\langle c\rangle \mid \bar{a}\langle b\rangle \mid b(y).P \rightarrow \bar{b}\langle c\rangle \mid b(y).P \rightarrow P[c/y]$$

then you get the π-calculus:

$P, Q :=$	$a(x).P,\ \ \bar{a}\langle v\rangle.P$	action prefix
	$P \mid Q,\ \ P + Q,\ \ 0$	composition, choice, arrêt
	$*P$	replicated process
	$(va)P$	name restriction

It is sometimes called *mobile processes* because the communication topology evolves during execution.

Proofs as programs

The *formulae as types* approach:

$$\begin{aligned} \text{formula} &\leftrightarrow \text{type} \\ \text{proof rules} &\leftrightarrow \text{primitive instructions} \\ \text{proof} &\leftrightarrow \text{program} \\ \text{normalization} &\leftrightarrow \text{evaluation} \end{aligned}$$

Works perfectly for λ-calculus and intuitionistic logic:

- *proof semantics* for LJ, AF2, Type theory...
- *program semantics* for PCF and extensions...
- *type systems* for programming languages...
- *operational insights* through game semantics, linear logic...

Proofs as programs – typed λ-calculi

- Computation: $(\lambda x.M)N \to_\beta M[N/x]$
- Typing: $\dfrac{}{\Gamma, x : A \vdash x : A}$ $\quad \dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$ $\quad \dfrac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash (M)N : B}$

Theorem

Reduction is conluent, typed terms are strongly normalising.

Many variations around this:

- extensions that preserve good properties (quantification, dependent types, etc.)
- restrictions (linearity, implicit complexity)
- inconsistent extensions (arbitrary fixed points, $D = D \to D$)

Proofs as programs – interaction and concurrency?

Intuitive adaptation of *formulae as types* for the concurrent world:

formula	$\leftrightarrow$	type	*but what is a type?*
proof rules	$\leftrightarrow$	primitive instructions	
proof	$\leftrightarrow$	process	
normalization	$\leftrightarrow$	execution	

The logic should feature

linearity : agents cannot be duplicated

symmetry : processes are not functions (rather relations)

Proofs as programs – Linear logic

Proofs in linear logic are strongly remeniscent of interacting processes:

- a cut is a connection between two parts of a system,
- cut elimination makes dual actions interact,
- quotienting proofs up to permutations of rules reveales parallelism in interaction and simplifies syntax, like structural congruence

It is tempting to interpret proofs as processes and transpose known tools from the λ-calculus into the concurrent world.

Samson Abramsky
Computational interpretations of linear logic
TCS 1993

First formulation of this idea, looks fine as a reading of cut-elimination in proof nets using vocabulary of process algebras.

Linear logic and the π-calculus

A speed run (don't worry with the details)

Axiom and cut for composition:

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

Actions:

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

Additive connectives for choice:

$$\frac{P \vdash \Gamma, \vec{x} : A \qquad Q \vdash \Gamma, \vec{y} : B}{P + Q \vdash \Gamma, \vec{x}\vec{y} : A \,\&\, B} \qquad \frac{P \vdash \Gamma, \vec{x} : A}{P \vdash \Gamma, \vec{x}\vec{y} : A \oplus B}$$

Multiplicative connectives for parallelism:

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

Exponentials for replication.

Linear logic and the π-calculus

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 π-calculus
LICS 2001

Independently developped:

Luís Caires and Frank Pfenning
Session types as intuitionistic linear propositions
Concur 2010

appears as very close.

linear logic and then π-calculus – the problems

Shortcomings:

- Typed processes are essentially functional.
- Many well-behaved interaction patterns are not typable.

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

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 state is less relevant,
- a process may behave differently depending on scheduling.

Some information is missing.

2

Logical description of processes

Proofs as schedules

The principles of our interpretation:

$$\begin{aligned} \text{formula} &\leftrightarrow \text{type of interaction} \\ \text{proof rules} &\leftrightarrow \text{primitives for building schedules} \\ \text{proof} &\leftrightarrow \text{schedule for a program} \\ \text{normalization} &\leftrightarrow \text{evaluation according to a schedule} \end{aligned}$$

This is not exactly:

- *Curry-Howard* for processes:
  proofs are not programs, but behaviours of programs
- *Proof search*:
  there is significant dynamics in cut-elimination

but a sort of middle ground in between.

EB and Virgile Mogbil
Proofs as executions
IFIP TCS 2012

Linear logic with named quantifiers

The formulas

Types of schedules:

$A, B :=$	$\exists_a \vec{x}.A$, $\forall_a \vec{x}.A$	send/receive some tuple then act as A
	$A \oplus B$, $A \,\&\, B$	choice between A and B (internal, external)
	$A \otimes B$, $A \parr B$	combination of A and B (indepent, correlated)
	$!A$, $?A$	repeated behaviour
	$\mathbf{1}$, $\bot$	empty behaviour (neutrals of $\otimes$ and $\parr$)
	$X\vec{v}$, $X\vec{v}^{\bot}$	type parameter

Negation $\sim A$ exchanges the left and right columns.

Interpretation of a typing judgement:

$\vdash P : A_1, \ldots, A_n$ — P *can* exhibit the behaviours A_i, they may be correlated

Typing rules by example – I

- Reception exactly corresponds to universal quantification:

$$\frac{\vdash P : \Gamma, A \qquad x \text{ does not occur in } \Gamma}{\vdash u(x).P : \Gamma, \forall_u x.A}$$

- Parallel composition is introduced as a conjunction:

$$\frac{\vdash P : \Gamma, A \qquad \vdash Q : \Delta, B}{\vdash P \mid Q : \Gamma, \Delta, A \otimes B} \qquad \frac{\vdash P : \Gamma, A, B}{\vdash P : \Gamma, A \parr B}$$

Typing rules by example – I

- Reception exactly corresponds to universal quantification:

$$\frac{\vdash P : \Gamma, A \qquad x \text{ does not occur in } \Gamma}{\vdash u(x).P : \Gamma, \forall_u x.A}$$

- Parallel composition is introduced as a conjunction:

$$\frac{\vdash P : \Gamma, A \qquad \vdash Q : \Delta, B}{\vdash P \mid Q : \Gamma, \Delta, A \otimes B} \qquad \frac{\vdash P : \Gamma, A, B}{\vdash P : \Gamma, A \parr B}$$

- Emission combines existential quantification and conjunction:

$$\frac{\vdash P : \Gamma, B}{\vdash \bar{u}\langle v \rangle.P : \Gamma, \exists_u x.{\sim}A(x), A(v) \otimes B}$$

"On u, I send some name x and expect a behaviour $A(x)$ from the receiver; besides, I expose $A(v)$ in parallel with B."

Typing rules by example – II

- A name can be made private if its behaviour is correct:

$$\frac{\vdash P : \Gamma, U \qquad u \notin \Gamma}{\vdash (\nu u)P : \Gamma} \quad \text{if} \quad U = \begin{cases} \exists_u x.{\sim}A \otimes \forall_u x.A & \text{linear usage} \\ ?\exists_u x.{\sim}A \otimes !\forall_u x.A & \text{client/server} \\ \ldots \text{ other possibilities } \ldots \end{cases}$$

Typing rules by example – II

- A name can be made private if its behaviour is correct:

$$\frac{\vdash P : \Gamma, U \qquad u \notin \Gamma}{\vdash (\nu u)P : \Gamma} \quad \text{if} \quad U = \begin{cases} \exists_u x.{\sim}A \otimes \forall_u x.A & \text{linear usage} \\ ?\exists_u x.{\sim}A \otimes !\forall_u x.A & \text{client/server} \\ \ldots \text{ other possibilities } \ldots \end{cases}$$

- Subtyping allows reasoning on behaviours:

$$\frac{\vdash P : \Gamma, A \qquad A \leq B}{\vdash P : \Gamma, B}\ \textsc{sub}$$

where behavioural subtyping $A \leq B$ is generated by rules like

$$A \otimes (B \parr C) \leq (A \otimes B) \parr C \qquad \text{semi-distributivity}$$

$$A \otimes \mathbf{1} \leq A \qquad \text{neutrality of } \mathbf{1}$$

$$?A \parr ?A \leq ?A \qquad \text{contraction}$$

... and possibly other logically sound implications ...

Typing vs reduction – an example

Let Ax be a predicate dependening on x, set $Bx := \forall_x z.Az$.

$$\dfrac{\dfrac{\dfrac{}{\vdash \bar{x}\langle a\rangle : {\sim}Bx, Aa} \quad \dfrac{\vdash P : Az}{\vdash y(z).P : By}}{\vdash \bar{x}\langle a\rangle \mid y(z).P : {\sim}Bx \otimes By, Aa}}{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) : \forall_u xy.({\sim}Bx \otimes By), Aa} \quad \dfrac{\dfrac{}{\vdash \bar{u}\langle bb\rangle : \exists_u xy.(Bx \parr {\sim}By), {\sim}Bb \otimes Bb}}{\vdash (vb)\bar{u}\langle bb\rangle : \exists_u xy.(Bx \parr {\sim}By)}$$

$$\dfrac{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid (vb)\bar{u}\langle bb\rangle : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa}{\vdash (vu)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid (vb)\bar{u}\langle bb\rangle) : Aa}$$

Typing vs reduction – an example

Let Ax be a predicate dependening on x, set $Bx := \forall_x z.Az$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \bar{x}\langle a\rangle : {\sim}Bx, Aa} \quad \dfrac{\vdash P : Az}{\vdash y(z).P : By}}{\vdash \bar{x}\langle a\rangle \mid y(z).P : {\sim}Bx \otimes By, Aa}}{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) : \forall_u xy.({\sim}Bx \otimes By), Aa} \quad \dfrac{}{\vdash \bar{u}\langle bb\rangle : \exists_u xy.(Bx \parr {\sim}By), {\sim}Bb \otimes Bb}}{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa, {\sim}Bb \otimes Bb}}{\vdash (vb)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle) : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa}}{\vdash (vu)(vb)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle) : Aa}$$

Typing vs reduction – an example

Let Ax be a predicate dependening on x, set $Bx := \forall_x z.Az$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \bar{x}\langle a\rangle : {\sim}Bx, Aa} \quad \dfrac{\vdash P : Az}{\vdash y(z).P : By}}{\vdash \bar{x}\langle a\rangle \mid y(z).P : {\sim}Bx \otimes By, Aa}}{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) : \forall_u xy.({\sim}Bx \otimes By), Aa} \quad \dfrac{}{\vdash \bar{u}\langle bb\rangle : \exists_u xy.(Bx \parr {\sim}By), {\sim}Bb \otimes Bb}}{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa, {\sim}Bb \otimes Bb}}{\vdash (vb)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle) : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa}}{\vdash (vu)(vb)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle) : Aa}$$

$$\longrightarrow \quad \dfrac{\dfrac{\dfrac{}{\vdash \bar{b}\langle a\rangle : {\sim}Bb, Aa} \quad \dfrac{\vdash P : Az}{\vdash b(z).P : Bb}}{\vdash \bar{b}\langle a\rangle \mid b(z).P : {\sim}Bb \otimes Bb, Aa}}{\vdash (vb)(\bar{b}\langle a\rangle \mid b(z).P) : Aa}$$

Typing vs reduction – an example

Let Ax be a predicate dependening on x, set $Bx := \forall_x z.Az$.

$$\dfrac{\dfrac{\dfrac{}{\vdash \bar{x}\langle a\rangle : {\sim}Bx, Aa} \quad \dfrac{\vdash P : Az}{\vdash y(z).P : By}}{\vdash \bar{x}\langle a\rangle \mid y(z).P : {\sim}Bx \otimes By, Aa}}{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) : \forall_u xy.({\sim}Bx \otimes By), Aa} \quad \dfrac{}{\vdash \bar{u}\langle bb\rangle : \exists_u xy.(Bx \parr {\sim}By), {\sim}Bb \otimes Bb}$$
$$\dfrac{\vdash u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa, {\sim}Bb \otimes Bb}{\vdash (vb)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle) : \forall_u xy.({\sim}Bx \otimes By) \otimes \exists_u xy.(Bx \parr {\sim}By), Aa}$$
$$\vdash (vu)(vb)(u(xy).(\bar{x}\langle a\rangle \mid y(z).P) \mid \bar{u}\langle bb\rangle) : Aa$$

$$\longrightarrow \quad \dfrac{\dfrac{}{\vdash \bar{b}\langle a\rangle : {\sim}Bb, Aa} \quad \dfrac{\vdash P : Az}{\vdash b(z).P : Bb}}{\vdash \bar{b}\langle a\rangle \mid b(z).P : {\sim}Bb \otimes Bb, Aa}$$
$$\vdash (vb)(\bar{b}\langle a\rangle \mid b(z).P) : Aa$$

$$\longrightarrow \quad \vdash P[a/z] : Aa$$

Soundness

This system admits two interpretations, depending on what we accept in the subtyping relation. Subtyping is

tight when $A \leq B$ implies that any action in A occurs in B:

- typability is preseved by reduction,
- typed processes cannot go wrong.

loose if it accepts propositional consequences like $A \otimes {\sim}A \leq \bot$:

- each typing induces an way of executing the term,
- each correct execution of a term can by transformed into a typing.

So “loose” implies “may”-type properties while “tight” implies “must”-type properties.

Proofs-as-schedules theorems

When using loose subtyping

Normal form: hiding is used only for sending fresh names (e.g. $(\nu x)\bar{u}\langle x\rangle$).

Theorem (soundness)

For each typing derivation of $\vdash P : \Gamma$ *there is an execution* $P \to^* P'$ *with* $\vdash P' : \Gamma$ *and* P' *is in* normal form.

Idea of the proof: perform cut elimination on the proof of $\vdash P : \Gamma$, read each elimination as a congruence or a reduction step of P.

Proofs-as-schedules theorems

When using loose subtyping

Normal form: hiding is used only for sending fresh names (e.g. $(\nu x)\bar{u}\langle x\rangle$).

Theorem (soundness)

For each typing derivation of $\vdash P : \Gamma$ *there is an execution* $P \to^* P'$ *with* $\vdash P' : \Gamma$ *and* P' *is in* normal form.

Idea of the proof: perform cut elimination on the proof of $\vdash P : \Gamma$, read each elimination as a congruence or a reduction step of P.

Theorem (completeness)

For all execution $P \to^* Q$ *where each name is used linearly, if* Q *is in normal form then* P *is typable.*

Idea of the proof: show the each normal form is typable and that each reduction step on linear names preserves typability backwards.

Type preservation

When using tight subtyping

Theorem

For all typed term $\vdash P : \Gamma$ *and reduction* $P \to Q$ *on a private name,* $\vdash Q : \Gamma$ *holds.*

Idea of the proof:

- Define a labelled transition system over types with rules like

$$\frac{}{\forall_u x.A \xrightarrow{u} \forall x.A} \qquad \frac{A \xrightarrow{\ell} A'}{A \otimes B \xrightarrow{\ell} A' \otimes B} \qquad \frac{A \xrightarrow{\bar{u}} F[\exists x.{\sim}C] \quad A \xrightarrow{u} G[\forall x.C]}{A \otimes B \xrightarrow{[u]} F[G[\bot]]}$$

  so that internal transitions do not change types,
- prove that a term is simulated by its type:
  if $P \xrightarrow{\ell} P'$ and $\vdash P : \Gamma$ then $\vdash P' : \Gamma'$ for some $\Gamma \xrightarrow{\ell} \Gamma'$.

Well typed programs cannot go wrong

When using tight subtyping

Theorem

If a typing $\vdash P : \Gamma$ *is derivable, then*

- P *has no infinite sequence of internal reductions,*
- *if* Γ *contains an action, then* P *has either an internal transition or an obervable transition described by* Γ*.*

Idea of the proof for termination:

- Define orthogonality as $P \perp\!\!\!\perp Q$ when $P \mid Q$ has no infinite sequence of internal transitions,
- interpret formulas as sets of processes with $[\![\sim A]\!] = [\![A]\!]^{\perp\!\!\!\perp}$ and

$$[\![1]\!] := \{0\}^{\perp\!\!\!\perp\perp\!\!\!\perp} \qquad [\![A \otimes B]\!] := \{(P \mid Q) \mid P \in [\![A]\!], Q \in [\![B]\!]\}^{\perp\!\!\!\perp\perp\!\!\!\perp}$$

$$[\![\forall_u \vec{x}.A]\!] := \left\{u.\lambda\vec{x}.P \;\middle|\; \forall \vec{v}, P[\vec{v}/\vec{x}] \in [\![A[\vec{v}/\vec{x}]]\!]\right\}...$$

- check that $[\![A]\!]$ cannot contain diverging processes,
- prove that $\vdash P : A$ implies $P \in [\![A]\!]$.

3

Semantic consequences

Semantics of proofs and processes

If formulas or proofs represent processes, then surely the semantics of proofs can provide models for processes in some sense. In the following:

- a brief presentation of the relational model of proofs,
- an implementation of it for our typing logic,
- some observations.

Semantics of proofs and processes

If formulas or proofs represent processes, then surely the semantics of proofs can provide models for processes in some sense. In the following:

- a brief presentation of the relational model of proofs,
- an implementation of it for our typing logic,
- some observations.

BEWARE
EXPERIMENTAL MATERIAL

The relational model – formulas

Each formula A is interpreted as a set $[\![A]\!]$:

$$[A \otimes B] = [A \parr B] = [A] \times [B],$$
$$[A \oplus B] = [A \,\&\, B] = \{0\} \times [A] \cup \{1\} \times [B],$$
$$[(\forall x \in E)A] = [(\exists x \in E)A] = \{(a, b) \mid a \in E, b \in [A[a/x]]\}$$

Interpretation of the points in $[\![A]\!]$:

- statically: bits of information about a value of type A
- dynamically: *experiments* for interacting with proofs of A

The relational model – formulas

Each formula A is interpreted as a set $[\![A]\!]$:

$$[A \otimes B] = [A \parr B] = [A] \times [B],$$
$$[A \oplus B] = [A \,\&\, B] = \{0\} \times [A] \cup \{1\} \times [B],$$
$$[(\forall x \in E)A] = [(\exists x \in E)A] = \{(a, b) \mid a \in E, b \in [A[a/x]]\}$$

Interpretation of the points in $[\![A]\!]$:

- statically: bits of information about a value of type A
- dynamically: *experiments* for interacting with proofs of A

Optionally: for each A, pose $[\![A]\!] \subseteq \mathscr{P}([A])$ a set of *valid* sets of points

- Coherence: for all $a \in [\![A]\!]$ and $b \in [\![{\sim}A]\!]$, require $\sharp(a \cap b) \leq 1$
- Finiteness: for all $a \in [\![A]\!]$ and $b \in [\![{\sim}A]\!]$, require that $a \cap b$ is finite

The relational model – proofs

A proof π of $\vdash A_1, ..., A_n$ is interpreted as a relation $[\![\pi]\!] \subseteq [A_1] \times ... \times [A_n]$ by annotating proofs with *experiments*:

$$\frac{\vdash \vec{v} : \Gamma, a : A \qquad \vdash \vec{w} : \Gamma, b : B}{\vdash \vec{v} : \Gamma, \vec{w} : \Gamma, (a, b) : A \otimes B} \qquad \frac{\vdash \vec{v} : \Gamma, a : A[b/x]}{\vdash \vec{v} : \Gamma, (b, a) : (\exists x \in E)A} \qquad ...$$

- The set $[\![\pi]\!]$ is preseved by cut elimination.
- The set $[\![\pi]\!]$ is *valid*: if $\pi \vdash A$ then $[\![\pi]\!] \in [\![A]\!]$
  (each notion of validity may accept additional logical rules)

Various information can be extracted from this kind of model.

Experiments as traces

Interpret named quantifiers as transition labels:

$$[\forall_u \vec{x}.A] = [\exists_u \vec{x}.A] = \left\{ u(\vec{x}).p \;\middle|\; p \in [A] \right\}$$

so we have points in a language of enriched traces:

$$\begin{array}{rll} p, q := & \ell.p & \text{perform transition } \ell \text{ then do } p \\ & (p, q) & \text{split into } p \text{ and } q \\ & (i, p) & \text{with } i \in \{0, 1\}\text{: choose side } i \text{ then do } p \end{array}$$

Experiments as traces

Interpret named quantifiers as transition labels:

$$[\forall_u \vec{x}.A] = [\exists_u \vec{x}.A] = \left\{ u(\vec{x}).p \;\middle|\; p \in [A] \right\}$$

so we have points in a language of enriched traces:

$$\begin{array}{rll} p, q := & \ell.p & \text{perform transition } \ell \text{ then do } p \\ & (p, q) & \text{split into } p \text{ and } q \\ & (i, p) & \text{with } i \in \{0, 1\}\text{: choose side } i \text{ then do } p \end{array}$$

- $p \in [\![\vdash P : \Gamma]\!]$ if the typed process P can exhibit the trace p
- extra validity conditions induced properties of interaction:
  - coherence $\longrightarrow$ interaction is deterministic
  - finiteness $\longrightarrow$ finitely branching non-determinism
- standard (sequential) traces can be deduced as interleavings of enriched traces