A decorated π-calculus for typing and cut elimination

Julien Gabet

Mars-June, 2018

Contents

Introduction

Theory of mobile processes is an important domain for research today, be it for its applications on network interactions (be it web services, the internet of things or other kinds of networks), or for its importance in logical foundations of computer science (proof nets, concurrency and continuation passing are the strong reference points in this domain). A usual theory of mobile processes is the π-calculus, on which a lot of work has been done and many open issues are still the topic of active research. A standard introduction to the topic is *The π-calculus, A Theory of Mobile Processes* by Sangiorgi and Walker [1].

Especially, the π-calculus as introduced in all generality is strongly non-deterministic in the reduction of its processes, which is a fundamental characteristic of the systems it is conceived to model. Many approaches exist for circumventing this difficulty, most of which involve developing type systems for the π-calculus that help induce some degree of determinism. Most of these systems only work for very small fragments of the π-calculus, often excluding some of its idioms, and there is much work to do to try and unify these type systems as well, though it is not the topic of this internship.

We will go the route of constructing a type system as well, using linear logic because its structure is very similar to the one of the terms of the π-calculus, which is a good start to thinking such a system would work well, and we will be adding annotations on terms as well to lie down the base structure. Prior work in that direction includes *On the π-Calculus and Linear Logic* by Bellin and Scott [2], *Linear Logic Propositions as Session Types* by Caires, Pfenning and Toninho [3] and *Linearity and the Pi-Calculus* by Kobayashi, Pierce and Turner [4] which lay some bases for this work, although this approach notably diverges from those presented in these references.

Our goal here is to type a fragment of the π-calculus with a fragment of linear logic, with room for extending this in the future to the rest of π-calculus and linear logic. We introduce an annotated calculus to allow for prescribing interactions, as well as facilitate the typing process, each construct being given a typing rule that revolves around the annotations it uses. The annotations allow to make the classic terms deterministic by showing at each step a reduction that would allow to continue further, and the typing part gives a guarantee that the reduction will not be blocked before the end.

To guide the rest of the work, we start from standard π-calculus, and upon building its decorated variant, we also build a projection from the annotated calculus to the classic one, and build our reduction and type systems with that projection in mind, so as to stay as faithful to the original calculus as possible.

Especially, we will be using the multiplicative parts of the π-calculus, and of linear logic (MLL), and we will prove that our reduction system is confluent, and that it preserves typing and has a property of cut elimination. All along, we check and prove that it projects well on usual reduction of classic π-calculus, to ensure it behaves correctly in regards to this base theory.

The first part revolves on defining the calculi and the projection, first by reminding the part of the classic π-calculus we will build upon, then defining the decorated calculus and the projection.

In the second part we define the reduction system, prove results about its confluence and show how it projects to the reduction system of the usual π-calculus.

The last part introduces the type system and show results such as type preservation for reduction and cut elimination.

1 The two languages

1.1 The classic π-calculus

Let us first remind the multiplicative part of the classic π-calculus and its constructs, as done in [1]:

Definition 1

Given a countable set of names $N := \{t, u, v, \ldots\}$, the grammar defining the terms of the multiplicative π-calculus is as follows:

$$\begin{array}{lll} P, Q ::= 0 \ ; \ P|Q & & \textit{null action and parallel construction} \\ \quad u(t).P \ ; \ \bar{u}(v).P & & \textit{sending and receiving names over a channel} \\ \quad \tau.P \ ; \ (\nu u)P & & \textit{internal action and name binding} \end{array}$$

We define the set of free names of a term P by induction on terms as follows:

$$\begin{aligned} fn(0) &= \emptyset \\ fn(P|Q) &= fn(P) \cup fn(Q) \\ fn(u(t).P) &= (fn(P) \backslash \{t\}) \cup \{u\} \\ fn(\bar{u}(v).P) &= fn(P) \cup \{u, v\} \\ fn(\tau.P) &= fn(P) \\ fn((\nu u)P) &= fn(P) \backslash \{u\} \end{aligned}$$

We also define a structural congruence on terms as follows:

$$\begin{aligned} P|0 &\equiv P \\ P|Q &\equiv Q|P \\ P|(Q|R) &\equiv (P|Q)|R \\ (\nu u)(\nu v)P &\equiv (\nu v)(\nu u)P \\ (\nu u)0 &\equiv 0 \\ (\nu u)(P|Q) &\equiv P|(\nu u)Q \quad \textit{if } u \notin fn(P) \end{aligned}$$

and a reduction relation given by the rules:

$$\frac{}{u(t).P|\bar{u}(v)Q \to P[v/t]|Q} \qquad \frac{}{\tau.P \to P}$$

$$\frac{P \to P'}{P|Q \to P'|Q} \qquad \frac{P \to P'}{(\nu u)P \to (\nu u)P'} \qquad \frac{P \equiv Q \to Q' \equiv P'}{P \to P'}$$

where the substitution is defined in an usual fashion by induction on terms.

The main mechanism here is the sending/receiving system that allows multiple agents to work in parallel and exchange data, this data consisting on names for further communication channels or or even addresses to other processes. This is what modelizes the network system, and this is the core of the non determinism this model allows to study.

With this system defined, we would like to look at a few examples:

Example: a simple process that one could imagine would be something like $\Big(u(t).P|Q\Big)|\bar{u}(v).R$.
This process needs the use of congruence to proceed, and has one reduction:

$$\begin{aligned} \Big(u(t).P|Q\Big)|\bar{u}(v).R &\equiv \Big(Q|u(t).P\Big)|\bar{u}(v).R \\ &\equiv Q|\Big(u(t).P|\bar{u}(v).R\Big) \\ &\to Q|\Big(P[v/t]|R\Big) \end{aligned}$$

Example: let's take a term which would want to communicate a private name over a public channel to some other process, in order to use this name as a new channel to communicate with this second process for the rest of

execution, especially waiting for a response on this name before continuing execution. Such a term can be of the form $(\nu v)(\bar{u}(v).\tau.0|v(t).P)$. It could then interact with anything of the form $u(t).(\bar{t}(w).\tau.0|Q)$.
Putting these together:

$$\begin{aligned}
(\nu v)\Big(\bar{u}(v).\tau.0|v(t').P\Big)|u(t).\Big(\bar{t}(w).\tau.0|Q\Big) &\equiv (\nu v)\Big(\Big(\bar{u}(v).\tau.0|v(t).P\Big)|u(t).\Big(\bar{t}(w).\tau.0|Q\Big)\Big)\\
&\equiv (\nu v)\Big(\Big(v(t').P|\bar{u}(v).\tau.0\Big)|u(t).\Big(\bar{t}(w).\tau.0|Q\Big)\Big)\\
&\equiv (\nu v)\Big(v(t').P|\Big(\bar{u}(v).\tau.0|u(t).\big(\bar{t}(w).\tau.0|Q\big)\Big)\Big)\\
&\to (\nu v)\Big(v(t').P|\Big(\tau.0|\big(\bar{v}(w).\tau.0|Q[v/t]\big)\Big)\Big)\\
&\to (\nu v)\Big(v(t').P|\Big(0|\big(\bar{v}(w).\tau.0|Q[v/t]\big)\Big)\Big)\\
&\equiv (\nu v)\Big(v(t').P||\Big(\bar{v}(w).\tau.0|Q[v/t]\Big)\Big)\\
&\equiv (\nu v)\Big(\Big(v(t').P|\bar{v}(w).\tau.0\Big)|Q[v/t]\Big)\\
&\to (\nu v)\Big(\Big(P[w/t']|\tau.0\Big)|Q[v/t]\Big)\\
&\to (\nu v)\Big(\Big(P[w/t']|0\Big)|Q[v/t]\Big)\\
&\equiv (\nu v)\Big(P[w/t']|Q[v/t]\Big)
\end{aligned}$$

We will follow this example in the paper, as it will be able to show a great variety of propositions applied to a practical example.

Remark: we see that the reduction $\to$ here is not confluent. A good example for that would be the term $(u(t).P|\bar{u}(v).Q)|u(t).S \equiv u(t).P|(\bar{u}(v).Q|u(t).S)$ that can reduce to either $(P[v/t]|Q)|u(t).S$ or $u(t).P|(Q|S[v/t])$, and those reduction paths do not have a common reduction in general, especially if P, Q and S are independent. That gives us a system that is highly non-deterministic when studied up to structural congruence. That could lead to terms with multiple reduction paths, but some of them could end up blocked at some point while others could end the reduction to the base term 0. This specific behavior with multiple reduction paths possibly being blocked is what we will try to avoid with the decorated calculus, by forcing a reduction path with annotations.

1.2 The annotated π-calculus and its projection

We now need to define the language of terms of our annotated π-calculus. We introduce the annotation with variables in order to identify what part of a term has to interact with what other part.

Definition 2

Take two countable sets, $N := \{t, u, v, \ldots\}$ the set of names as in the classic π-calculus, and $V := \{x, y, z, \ldots\}$ the set of variables for the decorations.
The annotated terms are defined by the following grammar:

$P, Q ::= x \leftrightarrow y \ ; \ 0_x \ ;$	$x, y \in V$ *base terms: equalizer and null action*
$\epsilon_x.P \ ;$	$x \notin fv(P)$ *variable introduction scheduling prefix*
$\lambda_x y.P \ ;$	$x, y \in fv(P)$ *variables binding scheduling prefix*
$u_x(t).P \ ; \ \bar{u}_x\langle v\rangle.P \ ;$	$t, u, v \in N, x \in fv(P)$ *action prefixes*
$P\|_x Q \ ; \ P\|\|_x Q \ ;$	$x \in fv(P) \cap fv(Q)$ *parallel and synchronization*
$(\nu u)P$	$u \in N$ *name binding prefix*

where construction $x \leftrightarrow y$ is commutative, and $fv(P)$ and $fn(P)$ are defined by induction on terms as follows:

$$
\begin{aligned}
fv(x \leftrightarrow y) &= \{x, y\} & fn(x \leftrightarrow y) &= \emptyset \\
fv(0_x) &= \{x\} & fn(0_x) &= \emptyset \\
fv(\epsilon_x.P) &= fv(P) \cup \{x\} & fn(\epsilon_x.P) &= fn(P) \\
fv(\lambda_x y.P) &= fv(P) \backslash \{y\} & fn(\lambda_x y.P) &= fn(P) \\
fv(u_x(t).P) &= fv(P) & fn(u_x(t).P) &= (fn(P) \backslash \{t\}) \cup \{u\} \\
fv(\bar{u}_x\langle v \rangle.P) &= fv(P) & fn(\bar{u}_x\langle v \rangle.P) &= fn(P) \cup \{u, v\} \\
fv((\nu u)P) &= fv(P) & fn((\nu u)P) &= fn(P) \backslash \{u\} \\
fv(P|_x Q) &= fv(P) \cup fv(Q) & fn(P|_x Q) &= fn(P) \cup fn(Q) \\
fv(P||_x Q) &= (fv(P) \cup fv(Q)) \backslash \{x\} & fn(P||_x Q) &= fn(P) \cup fn(Q)
\end{aligned}
$$

Remark: the only construction with a prefix that adds to the variables of a term is the variable introduction (ϵ_x construction). That is because, as the rules are defined here for the other prefixes, the variable used to decorate the prefix already exists in the subterm, and the variable is used as a follow-up for the behavior of this term. The λ is also binding for its second variable (y here), keeping availability for its first variable only. The ϵ, on the opposite, creates its variable, that does not exist in the base term.

Remark: in some constructions and some cases, it can be useful to have a notation to describe in which subterm a variable appears. We will choose for that to put the variable in exponent, so that $(P^x|_y Q)|_x R^x$ denotes the fact that x appears in P and R, but not in Q.

Remark: we also define substitution of names and variables by induction on the terms the exact same way as usual. Though, it is important to note that, because names and variables are two distinct and independent sets, substituting a variable for a name (or the opposite) is not possible. The distinction between those two sets is important for the reduction to pose no problem of variable capture, and allows for something akin to α-conversion from λ-calculi: one can rename freely all instances of a free variable or name in a term for a fresh one, or all instances of a bound variable or name (including the binding operator) for a fresh one as well *in the scope of the binding operator*, and the term is still considered the same.

This calculus has very similar constructions to the classic π-calculus defined above. Of note though is that the parallel operation, the internal action and the null process have two equivalent each in the decorated calculus. For the parallel operation, the meaning is that it puts two processes together, but they might not interact with each other, which is why the variable used to observe them together is kept available to the outside world, while the synchronization puts the processes in interaction with each other on this variable, which is why it is hidden from the exterior, acting as a binding operation. The ϵ is an internal action that creates a variable to observe on, while the λ is an internal action that puts two variables together to observe as one. The 0 is still an inactive process, which we do not need to observe on more than one variable, and the equalizer is a bit more complex, as it does nothing observable but create two observation variables. This specificity will be addressed in the projection:

Definition 3

We define a projection operator $\lfloor \cdot \rfloor$ from the annotated calculus to the classic π-calculus as follows:

$$
\begin{aligned}
\lfloor 0_x \rfloor &= 0 \\
\lfloor x \leftrightarrow y \rfloor &= \tau.0 \\
\lfloor \lambda_x y.P \rfloor &= \tau.\lfloor P \rfloor \\
\lfloor \epsilon_x.P \rfloor &= \tau.\lfloor P \rfloor \\
\lfloor u_x(t).P \rfloor &= u(t).\lfloor P \rfloor \\
\lfloor \bar{u}_x\langle v \rangle.P \rfloor &= \bar{u}(v).\lfloor P \rfloor \\
\lfloor P|_x Q \rfloor &= \lfloor P \rfloor | \lfloor Q \rfloor \\
\lfloor P||_x Q \rfloor &= \lfloor P \rfloor | \lfloor Q \rfloor \\
\lfloor (\nu u)P \rfloor &= (\nu u)\lfloor P \rfloor
\end{aligned}
$$

Remark: The way we address the complicated meaning of the equalizer is by projecting it to $\tau.0$, which is a bit hacky, but is justified later by allowing for the reduction we want to define on the annotated calculus to project exactly on the reduction of the classic π-calculus, and because the equalizer will be associated with the axiom rule in the type system, which means it cannot be used as a prefix like ϵ or λ. That construction done now allows all else to go well, and since there is no way to clearly observe a difference in behavior between 0 and $\tau.0$, especially since none of them interact with anything else, it is not too much of a problem to do that now, while it would create discrepancies later to not do it.

Example: let's see how our first example goes with this new system. What we want is to follow the projection backwards to create an annotated term. Obviously, there will be choices to be made as to whether we want to use a parallel or a synchronization for the pre-image of the parallel in the projection, as well as choices for the τ. One example of term that could work, and that we will be keeping for this paper, is the following:

$$(\nu v)\Big(\bar{u}_x\langle v\rangle.(x \leftrightarrow y)||_y v_y(t').P\Big)||_x u_x(t).\Big(\bar{t}_x\langle w\rangle.(x \leftrightarrow z)||_z Q\Big)$$

Why we chose this form of the term is both for having a nice term for later when we introduce the reduction rules, as well as because the two name interactions and the two internal actions we saw reduce in the first example were separate, hinting at something to do with the term itself and not its potential interactions with the exterior.

2 The execution of the decorated calculus

2.1 The reduction relation

The synchronization rule will be used to guide the behavior of terms, thus we define a reduction system over this construction specifically, as follows:

Definition 4

We define a reduction rule for synchronization (that does not hold under prefixes aside from ν) as follows:

$$\epsilon_x.P||_x 0_x \to P \qquad \text{symmetric in } || \quad 1$$
$$P||_x x \leftrightarrow y \to P[y/x] \qquad \text{symmetric in } || \quad 2$$
$$(P|_x Q)||_x \lambda_x y.R \to P||_x (Q[y/x]||_y R) \qquad 3a$$
$$\lambda_x y.R||_x (P|_x Q) \to (R||_x P)||_y Q[y/x] \qquad 3b$$
$$\bar{u}_x\langle v\rangle.P||_x u_x(t).Q \to P||_x Q[v/t] \qquad \text{symmetric in } || \quad 4$$

We define rules for commuting synchronization with parallel and ν when acceptable. This happens inside contexts, especially under prefixes:

$$(P|_x Q)||_y R \succ (P||_y R)|_x Q \qquad \text{symmetric in } ||, y \notin fv(Q) \quad 5a$$
$$(P|_x Q)||_y R \succ P|_x (Q||_y R) \qquad \text{symmetric in } ||, y \notin fv(P) \quad 5b$$
$$(\nu u)P||_x Q \succ (\nu u)(P||_x Q) \qquad \text{symmetric in } ||, u \notin fn(Q) \quad 6$$

Remark: in some rules, we have strong requirements, such as a variable or a name not being present in one or more subterms. Failure to meet those requirements can lead to terms that do not reduce past a certain point, because they cannot be matched with a congruent one that reduces.

Example: the term $(x \leftrightarrow y|_x x \leftrightarrow y)||_y 0_y$ cannot reduce in any way, since no rule apply as y is present everywhere. The term $(x \leftrightarrow y|_x x \leftrightarrow z)||_y 0_y$ reduces though, first with rule 5a to $(x \leftrightarrow y||_y 0_y)|_x x \leftrightarrow z$ and then reducing with rule 2 in $0_x|_x x \leftrightarrow z$.

Definition 5

We define a congruence rule with the following axioms:

$$(P|_x Q)|_y R \equiv (P|_y R)|_x Q \qquad y \notin fv(Q), x \notin fv(R)$$
$$P|_x (Q|_y R) \equiv Q|_y (P|_x R) \qquad y \notin fv(P), x \notin fv(Q)$$
$$(P|_x Q)|_y R \equiv P|_x (Q|_y R) \qquad y \notin fv(P), x \notin fv(R)$$
$$(\nu u)(\nu v)P \equiv (\nu v)(\nu u)P$$
$$(\nu u)0_x \equiv 0_x$$
$$(\nu u)P|_x Q \equiv (\nu u)(P|_x Q) \qquad \text{symmetric in } |, u \notin fn(Q)$$

This rule is defined as an equivalence (reflexive, symmetric and transitive), and applies inside contexts, just like structural congruence does on the classic calculus.

Remark: the example from the first part is irreducible in its state for now. Indeed, we would want the synchronization to be able to commute with prefixes, but it does not do that, because we want the system to stay close to the actual execution of processes. An other way could be to allow commuting synchronization rules with each other when they clash. We could do that, but then we would lose a property of this system which is that it terminates, which is important later.

We would like these arrows to have a confluent behavior. That is why the congruence was needed, as we will see now in the next part.

2.2 Confluence property for $\to$ and $\succ$

Proposition 6

Relation $\to$ is strongly confluent.

▷ Since a synchronization makes the observed variable unavailable to the term, and because $\to$ does not allow for an arbitrary term on any side but only explicitly fixed constructions, and not allowing for these constructions to be a synchronization, we can remark that two reductions using $\to$ in the same term cannot interfere with each other. Thus, a term capable of two reductions using $\to$ has one of them be either a strict subterm of the other, or distinct parts of a more general term. The reductions then do not interfere with each other, and can be followed in any order. Only exception to that is the interaction of two equalizers synchronized on the same variable, but the commutativity of this construction makes it close immediately:

$$\begin{array}{ccccc} & & x \leftrightarrow y||_x x \leftrightarrow z & & \\ & \swarrow & & \searrow & \\ (x \leftrightarrow z)[y/x] & & & & (x \leftrightarrow y)[z/x] \\ \| & & & & \| \\ y \leftrightarrow z & & = & & z \leftrightarrow y \end{array}$$

□

Because of the specificity of the constructions in the $\to$, it does not directly interact with $\succ$ rules either, except for the $x \leftrightarrow y$ case.

We remind the definitions of a simulation and a bisimulation, as those tools are uncommon enough to justify being introduced here:

Definition 7

- *A simulation $\mathcal{R}$ is a binary relation on terms such that, for all terms P, Q, if $P\mathcal{R}Q$ then forall $P \to P'$ there exists a term Q' such that $Q \to Q'$ and $P'\mathcal{R}Q'$.*
- *A bisimulation $\mathcal{R}$ is a binary relation on terms such that $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations.*

Then, because of the lack of direct interaction between $\to$ and $\succ$, we have:

Proposition 8

The reflexive closure of relation $\succ$ is a simulation for $\to$.

▷ The cases to treat are those where $x \leftrightarrow y$ interacts with something. We treat them below, and also put the cut variable in exponent of the term where it appears in, in order to apply $\succ$ efficiently, where applicable:

$$\begin{array}{ccc} (P^y|_x Q)||_y x \leftrightarrow y \to (P^y|_x Q)[x/y] = P[x/y]|_x Q & & (\nu u)P||_z z \leftrightarrow x \longrightarrow ((\nu u)P)[x/z] = (\nu u)P[x/z] \\ \curlyvee_{5a} \quad \nearrow & & \curlyvee_{6} \quad \nearrow \\ (P^y||_y x \leftrightarrow y)|_x Q & & (\nu u)(P||_z z \leftrightarrow x) \end{array}$$

All other cases act the same way as for $\to$ not interacting with itself because one of the reductions occurs either in a strict subterm of the other, or in a distinct term from the other. □

The $\succ$ relation in itself has a bit more complexity to it being confluent, as two instances can interact with each other. The congruence is needed to make this relation confluent.

Proposition 9

Relation $\succ$ is confluent up to $\equiv$,
ie. for all $P_1 \prec P \succ P_2$, there exists P_1', P_2' such that

$$\begin{array}{ccc} & P & \\ \swarrow & & \searrow \\ P_1 & & P_2 \\ \curlyvee & & \curlyvee \\ P_1' & \equiv & P_2' \end{array}$$

▷ There are 3 possible cases here, for two groups of rules in the $\succ$ relation.
Rule 5 against rule 5: there are 4 cases, depending on where the variable cut against is situated. The variable is noted as an exponent in the terms where it appears, and only 2 cases are treated (as the other two are their complements, and are treated the same):

$$\begin{array}{ccc} & (P^y|_x Q)||_y (R^y|_z S) & \\ \swarrow_{5a,\,left} & & \searrow_{5a,\,right} \\ (P^y||_y (R^y|_z S))|_x Q & & ((P^y|_x Q)||_y R^y)|_z S \\ \curlyvee_{5a} & & \curlyvee_{5a} \\ ((P^y||_y R^y)|_z S)|_x Q & \equiv & ((P^y||_y R^y)|_x Q)|_z S \end{array}$$

$$\begin{array}{ccc} & (P^y|_x Q)||_y(R|_z S^y) & \\ \swarrow_{5a,\,left} & & \searrow_{5b,\,right} \\ (P^y||_y(R|_z S^S))|_x Q & & R|_z((P^y|_x Q)||_y S^y) \\ \downarrow_{5b} & & \downarrow_{5a} \\ (R|_z(P^y||_y S^y))|_x Q & \equiv & R|_z((P^y||_y S^y)|_x Q) \end{array}$$

Rule 6 against rule 6 works the same:

$$\begin{array}{ccc} & (\nu u)P||_y(\nu v)Q & \\ \swarrow_6 & & \searrow_6 \\ (\nu u)(P||_y(\nu v)Q) & & (\nu v)((\nu u)P||_y Q) \\ \downarrow_6 & & \downarrow_{6a} \\ (\nu u)(\nu v)(P||_y Q) & \equiv & (\nu v)(\nu u)(P||_y Q) \end{array}$$

The last set of cases is a rule 5 against rule 6. Those are all treated the same way as well, so we only treat one example:

$$\begin{array}{ccc} & (P^y|_x Q)||_y(\nu u)R & \\ \swarrow_{5a} & & \searrow_6 \\ (P^y||_y(\nu u)R)|_x Q & & (\nu u)((P^y|_x Q)||_y R) \\ \downarrow_6 & & \downarrow_{5a} \\ (\nu u)(P^y||_y R)|_x Q & \equiv & (\nu u)((P^y||_y R)|_x Q) \end{array}$$

That ends the proof of confluence for $\succ$ up to $\equiv$. □

We need $\equiv$ to not alter the behavior of $\to$, but here we do not have a simulation or bisimulation. Instead, we observe a phenomenon that gets close enough:

Proposition 10

For all $P \equiv Q$ and $P \to P'$,
there exists $Q \succ^* Q' \to Q'' \equiv P'' \prec^* P'$,

$$\begin{array}{ccc} P & \equiv & Q \\ \downarrow & & \downarrow^* \\ P' & & Q' \\ \downarrow^* & & \downarrow \\ P'' & \equiv & Q'' \end{array}$$

ie.

▷ We treat some interesting cases here, still where necessary with the cut variable in exponent where it appears:

$$\begin{array}{ccc} \epsilon_z.((P|_x Q)|_y R)||_z 0_z & \equiv & \epsilon_z.(P|_x(Q|_y R))||_z 0_z \\ \downarrow & & \downarrow \\ (P|_x Q)|_y R & \equiv & P|_x(Q|_y R) \end{array}$$

$$\begin{array}{ccc} ((P|_x Q^y)|_y R^y)||_y \lambda_y z.S & \equiv & (P|_x(Q^y|_y R^y))||_y \lambda_y z.S^y \\ \downarrow & & \downarrow \\ (P|_x Q^y)||_y(R[z/y]||_z S) & & P|_x((Q^y|_y R^y)||_y \lambda_y z.S^y) \\ & \searrow & \downarrow \\ & & P|_x(Q^y||_y(R[z/y]||_z S^y)) \end{array}$$

where $P \equiv P'$:

$$\begin{array}{ccc} u_x(t).P||_x \bar{u}_x\langle v\rangle.Q & \equiv & u_x(t).P'||_x \bar{u}_x\langle v\rangle.Q \\ \downarrow & & \downarrow \\ P[v/t]||_x Q & \equiv & P'[v/t]||_x Q \end{array}$$

□

This behavior is interesting because the diagram still closes, by allowing $\succ$ to act where needed. That gives us something very close to a simulation, but not quite. Something to study later could be to get a good relation formed by aggregating $\to$ and $\succ$ in a good way to have a simulation, or find an other way to reduce and a new congruence that is a simulation and has the other properties we want and will show with the typing part later.
For now let's see what this reduction system gives with the projection.

2.3 Projection on classic execution

The reduction here was formed to mimic as much as possible the one from the classic calculus. As such, it is not shocking to have a property like:

Proposition 11

If $P \to P'$ in the annotated calculus, then $\lfloor P \rfloor \to \lfloor P' \rfloor$ in usual π-calculus.
If $P \succ P'$ or $P \equiv P'$ in the annotated calculus, then $\lfloor P \rfloor \equiv \lfloor P' \rfloor$ in the usual π-calculus.

$\triangleright$ We only treat a few interesting cases, as all of them are almost immediate and writing them all would take place and not be of much interest:

$$\begin{array}{ccccc} \lambda_x y.R||_x(P|_x Q) & \longrightarrow & (R||_x P)||_y Q[y/x] \\ \lfloor\ \rfloor & & \lfloor\ \rfloor \\ \tau.R|(P|Q) & \longrightarrow R|(P|Q) & \equiv & (R|P)|Q \end{array} \qquad \begin{array}{ccc} \bar{u}_x\langle v\rangle.P||_x u_x(t).Q & \to & P||_x Q[v/t] \\ \lfloor\ \rfloor & & \lfloor\ \rfloor \\ \bar{u}(v).P|u(t).Q & \longrightarrow & P|Q[v/t] \end{array}$$

$$\begin{array}{ccccc} (P|_x Q)||_y R & \succ & P|_x(Q||_y R) \\ \lfloor\ \rfloor & & \lfloor\ \rfloor \\ (P|Q)|R & \equiv & P|(Q|R) \end{array} \qquad \begin{array}{ccc} (\nu u)P|_x Q & \equiv & (\nu u)(P|_x Q) \\ \lfloor\ \rfloor & & \lfloor\ \rfloor \\ (\nu u)P|Q & \equiv & (\nu u)(P|Q) \end{array}$$

□

Example: take the simple term we observed earlier in this part:

$$\begin{array}{ccccccc} (x \leftrightarrow y|_x x \leftrightarrow z)||_y 0_y & \succ & (x \leftrightarrow y||_y 0_y)|_x x \leftrightarrow z & \longrightarrow & 0_x|_x x \leftrightarrow z \\ \lfloor\ \rfloor & & \lfloor\ \rfloor & & \lfloor\ \rfloor \\ (\tau.0|\tau.0)|0 & \equiv & (\tau.0|0)|\tau.0 & \equiv & \tau.0|\tau.0 \longrightarrow 0|\tau.0 \end{array} .$$

3 Typing decorated terms with MLL

3.1 The typing system

We first define the language of formulas that we will use for the typing system:

Definition 12

The language of formulas is given by the following grammar:

$$\begin{array}{rll} A, B ::= & 1 \mid \bot & \text{neutrals} \\ & \mid A \otimes B \mid A \parr B & \text{tensor and par} \\ & \mid \exists_u t.A \mid \forall_u t.A & t, u \in N \quad \text{existential and universal quantifiers} \end{array}$$

It is based on the structure of MLL, and first order quantification annotated by those names and quantifying on those names. With that, we give a typing system in the form of deduction rules, of which the first 6 are exactly the rules for MLL if considering only the formulas in the logic part of the rules:

Definition 13

The typing system is given by the following rules, where Γ, Δ are partial functions from V the set of variables to the language of formulas, and where Γ and Δ do not share any variable name when they occur as two hypothesis of the same rule (PARA and CUT):

Rules for neutral elements:

$$\frac{}{0_x \vdash x : 1}\ NOP \qquad \frac{P \vdash \Gamma \quad x \notin \Gamma}{\epsilon_x.P \vdash \Gamma, x : \bot}\ BOT \qquad \frac{}{x \leftrightarrow y \vdash x : E^\bot, y : E}\ AX$$

Construction rules:

$$\frac{P \vdash \Gamma, x : E \quad Q \vdash \Delta, x : F}{P|_x Q \vdash \Gamma, \Delta, x : E \otimes F}\ PARA \qquad \frac{P \vdash \Gamma, x : E, y : F}{\lambda_x y.P \vdash \Gamma, x : E \parr F}\ LAM \qquad \frac{P \vdash \Gamma, x : E \quad Q \vdash \Delta, x : E^\bot}{P||_x Q \vdash \Gamma, \Delta}\ CUT$$

Action rules:

$$\frac{P \vdash \Gamma, x : A[v/t]^\bot}{\bar{u}_x\langle v\rangle.P \vdash \Gamma, x : \exists_u t.A^\bot}\ IN \qquad \frac{P \vdash \Gamma, x : A \quad t \notin \Gamma}{u_x(t).P \vdash \Gamma, x : \forall_u t.A}\ OUT$$

Nu rule:

$$\frac{P \vdash \Gamma \quad u \text{ does not appear in any formula of } \Gamma}{(\nu u)P \vdash \Gamma}\ NU$$

One important thing to note here, as explained before, is that for PARA and CUT rules, Γ and Δ are disjoint. The other important thing is that the synchronization construction becomes a CUT operation in the typing system. This rule has elimination transformations, given by the relations $\to$ and $\succ$ defined in 4. Properties we would like to emerge from such a construction would be that the arrows preserve typing, and that their application for cut elimination terminates and actually eliminates cuts.
Before that, let's look back at our example from part 1, which we can now type:

Example: The first part of the term is of the form:

$$\dfrac{\dfrac{x \to y \vdash x : \forall_v t'.A, y : \exists_v t'.A^\bot}{\bar{u}_x\langle v\rangle.(x \to y) \vdash x : \exists_u t.\forall_t t'.A, y : \exists_v t'.A^\bot} \quad \dfrac{\dfrac{(\pi_1)}{P \vdash \Gamma, y : A \quad t' \notin \Gamma}}{v_y(t').P \vdash \Gamma, y : \forall_v t'.A}}{\dfrac{\bar{u}_x\langle v\rangle.(x \to y)||_y v_y(t').P \vdash \Gamma, x : \exists_u t.\forall_t t'.A}{(\nu v)(\bar{u}_x\langle v\rangle.(x \to y)||_y v_y(t').P) \vdash \Gamma, x : \exists_u t.\forall_t t'.A}}\ \text{cut}$$

and the second part is of the form:

$$\dfrac{\dfrac{x \to z \vdash x : A[w/t']^\perp, z : A[w/t'] \qquad (\pi_2)}{\bar{t}_x\langle a\rangle.(x \to z) \vdash x : \exists_t t'.A^\perp, z : A[w/t'] \qquad Q \vdash \Delta, z : A[w/t']^\perp}\ \text{cut}}{\dfrac{\bar{t}_x\langle w\rangle.(x \to z)||_z Q \vdash \Delta, x : \exists_t t'.A^\perp \quad t \notin \Delta}{u_x(t).(\bar{t}_x\langle w\rangle.(x \to z)||_z Q) \vdash \Delta, x : \forall_u t.\exists_t t'.A^\perp}}$$

Then we put them together:

$$\dfrac{(\nu v)(\bar{u}_x\langle v\rangle.(x \to y)||_y v_y(t').P) \vdash \Gamma, x : \exists_u t.\forall_t t'.A \qquad u_x(t).(\bar{t}_x\langle w\rangle.(x \to z)||_z Q) \vdash \Delta, x : \forall_u t.\exists_t t'.A^\perp}{(\nu v)(\bar{u}_x\langle v\rangle.(x \to y)||_y v_y(t').P)||_x u_x(t).(\bar{t}_x\langle w\rangle.(x \to z)||_z Q) \vdash \Gamma, \Delta}\ \text{cut}$$

3.2 Type preservation

Proposition 14

Relations $\to$, $\succ$ and $\equiv$ preserve typing, ie. for all $P \vdash \Gamma$ provable in the typing system and there exists P' such that either $P \to P'$, $P \succ P'$ or $P \equiv P'$ can be derived in the reduction system, $P' \vdash \Gamma$ is provable.

▷ Let us treat a few cases, the most interesting ones, as the others work the same:

$$\dfrac{\dfrac{P \vdash \Gamma, x : A[v/t]^\perp}{\bar{u}_x\langle v\rangle.P \vdash \Gamma, x : \exists_u t.A^\perp} \quad \dfrac{Q \vdash \Delta, x : A \quad t \notin \Delta}{u_x(t).P \vdash \Delta, x : \forall_u t.A}\ \text{cut}}{\bar{u}_x\langle v\rangle.P||_x u_x(t).Q \vdash \Gamma, \Delta}\ \text{cut} \to \dfrac{P \vdash \Gamma, x : A[v/t]^\perp \qquad Q[v/t] \vdash \Delta, x : A[v/t] \quad t \notin \Delta}{P||_x Q[v/t] \vdash \Gamma, \Delta}\ \text{cut}$$

$$\dfrac{\dfrac{R \vdash \Gamma, x : A^\perp, y : B^\perp}{\lambda_x y.R \vdash \Gamma, x : A^\perp \parr B^\perp} \quad \dfrac{P \vdash \Delta, x : A \qquad Q \vdash \Theta, x : B}{P|_x Q \vdash \Delta, \Theta, x : A \otimes B}\ \text{cut}}{\lambda_x y.R||_x (P|_x Q) \vdash \Gamma, \Delta, \Theta}\ \text{cut} \to \dfrac{\dfrac{R \vdash \Gamma, x : A^\perp, y : B^\perp \qquad P \vdash \Delta, x : A}{R||_x P \vdash \Gamma, \Delta, y : B^\perp}\ \text{cut} \qquad Q[y/x] \vdash \Theta, y : B}{(R||_x P)||_y Q[y/x] \vdash \Gamma, \Delta, \Theta}\ \text{c}$$

$$\dfrac{\dfrac{P \vdash \Gamma, x : A, y : C \qquad Q \vdash \Delta, x : B}{P|_x Q \vdash \Gamma, \Delta, x : A \otimes B, y : C} \qquad R \vdash \Theta, y : C^\perp}{(P|_x Q)||_y R \vdash \Gamma, \Delta, \Theta, x : A \otimes B}\ \text{cut} \succ \dfrac{\dfrac{P \vdash \Gamma, x : A, y : C \qquad R \vdash \Theta, y : C^\perp}{P||_y Q \vdash \Gamma, \Theta, x : A} \qquad Q \vdash \Delta, x : B}{(P||_y R)|_x Q \vdash \Gamma, \Delta, \Theta, x : A \otimes B}$$

$$\dfrac{\dfrac{P \vdash \Gamma, x : A \quad u \notin \Gamma \cup \{A\}}{(\nu u)P \vdash \Gamma} \qquad Q \vdash \Delta, x : B \quad u \notin \Delta \cup \{B\}}{(\nu u)P|_x Q \vdash \Gamma, \Delta, x : A \otimes B} \equiv \dfrac{P \vdash \Gamma, x : A \quad u \notin \Gamma \cup \{A\} \qquad Q \vdash \Delta, x : B \quad u \notin \Delta \cup \{B\}}{P|_x Q \vdash \Gamma, \Delta \quad u \notin \Gamma \cup \Delta \cup \{A, B\}}$$
$$(\nu u)(P|_x Q) \vdash \Gamma, \Delta, x : A \otimes B$$

□

This property gives us something very important, as this means our reduction system is correct with the typing. That means well-typed terms do reduce to well-typed terms, and as such execution steps do not give ill-behaving terms.

Proposition 15

The rewriting system induced by $\to$ and $\succ$ terminates, ie. there is no infinite, strictly decreasing sequence of reductions for this rule.

▷ Let us again treat the same few cases, the most interesting ones, as the others work the same.
The decreasing measure is the sum of the sizes of the terms right under the cut:

$$\dfrac{\dfrac{P \vdash \Gamma, x : A[v/t]^\perp}{\bar{u}_x\langle v\rangle.P \vdash \Gamma, x : \exists_u t.A^\perp} \quad \dfrac{Q \vdash \Delta, x : A \quad t \notin \Delta}{u_x(t).P \vdash \Delta, x : \forall_u t.A}\ \text{cut}}{\bar{u}_x\langle v\rangle.P||_x u_x(t).Q \vdash \Gamma, \Delta}\ \text{cut} \to \dfrac{P \vdash \Gamma, x : A[v/t]^\perp \qquad Q[v/t] \vdash \Delta, x : A[v/t] \quad t \notin \Delta}{P||_x Q[v/t] \vdash \Gamma, \Delta}\ \text{cut}$$

Where both sides had their sizes reduced by 1, so the sum was reduces by 2.

$$\dfrac{\dfrac{R \vdash \Gamma, x : A^{\perp}, y : B^{\perp}}{\lambda_x y.R \vdash \Gamma, x : A^{\perp} \parr B^{\perp}} \quad \dfrac{P \vdash \Delta, x : A \qquad Q \vdash \Theta, x : B}{P|_x Q \vdash \Delta, \Theta, x : A \otimes B}}{\lambda_x y.R||_x (P|_x Q) \vdash \Gamma, \Delta, \Theta}\ \text{cut} \quad \rightarrow \quad \dfrac{\dfrac{R \vdash \Gamma, x : A^{\perp}, y : B^{\perp} \qquad P \vdash \Delta, x : A}{R||_x P \vdash \Gamma, \Delta, y : B^{\perp}} \text{cut} \qquad Q[y/x] \vdash \Theta, y : B}{(R||_x P)||_y Q[y/x] \vdash \Gamma, \Delta, \Theta}\ \text{c}$$

Where there is no more λ on R. Especially, the inner cut has size $|R| + |P| < |R| + 1 + |P| + |Q|$ and the outer one has size $|R| + |P| + |Q| < |R| + 1 + |P| + |Q|$.

$$\dfrac{\dfrac{P \vdash \Gamma, x : A, y : C \qquad Q \vdash \Delta, x : B}{P|_x Q \vdash \Gamma, \Delta, x : A \otimes B, y : C} \qquad R \vdash \Theta, y : C^{\perp}}{(P|_x Q)||_y R \vdash \Gamma, \Delta, \Theta, x : A \otimes B}\ \text{cut} \succ \dfrac{\dfrac{P \vdash \Gamma, x : A, y : C \qquad R \vdash \Theta, y : C^{\perp}}{P||_y Q \vdash \Gamma, \Theta, x : A} \qquad Q \vdash \Delta, x : B}{(P||_y R)|_x Q \vdash \Gamma, \Delta, \Theta, x : A \otimes B}$$

Where the cut is now above one of the subterms, especially its measure is now $|P| + |R| < |P| + |Q| + |R|$ □

Remark: This system preserves typing and terminates, but does not completely eliminate cut. For example, terms like $u_x(t).\epsilon_y.P||_y 0_y$ or the one from the example in part 1 cannot reduce, yet there are cuts not eliminated. Likewise to the general β-reduction that can reduce under λ in the λ-calculus, we need to extend relations $\rightarrow$ and $\succ$ to be able to reduce under action prefixes, and commute them with cut, as well as extend $\equiv$ to allow commuting action prefixes with other prefixes and parallel as well, when the variables of the operators being commuted do not clash.

3.3 Extending the reduction relations, and cut elimination

Definition 16

Relation $\rightarrow$ is extended into $\rightsquigarrow$ by allowing it to act under prefixes (no specific rule added).

We also extend $\succ$ into $\succsim$ as follows, allowing for it to act on (and under) prefixes:

$\epsilon_x.P\|\|_y Q \succsim \epsilon_x.(P\|\|_y Q)$	*symmetric in* $\|\|$, $x \notin fv(Q)$	6a
$\lambda_x y.P\|\|_z Q \succsim \lambda_x y.(P\|\|_z Q)$	*symmetric in* $\|\|$, $y \notin fv(Q)$	6b
$u_x(t).P\|\|_y Q \succsim u_x(t).(P\|\|_y Q)$	*symmetric in* $\|\|$	6c
$\bar{u}_x\langle v\rangle.P\|\|_y Q \succsim \bar{u}_x\langle v\rangle.(P\|\|_y Q)$	*symmetric in* $\|\|$	6d

We finally extend $\equiv$ into $\cong$ by adding the following rules:

$P\|_x \alpha_y.Q \cong \alpha_y.(P\|_x Q)$	*symmetric in* $\|, \alpha\cdot \in \{\epsilon., \lambda.z, u.(t), \bar{u}.\langle v\rangle\}, y, t \notin fv(P)$
$\alpha_x.\beta_y.P \cong \beta_y.\alpha_x.P$	$\alpha., \beta. \in \{\epsilon., \lambda.z, u.(t), \bar{u}.\langle v\rangle\}, x \neq y$
$(\nu u)\alpha_x.P \cong \alpha_x.(\nu u)P$	$\alpha. \in \{\epsilon., \lambda.z, v.(t), \bar{v}.\langle w\rangle, u \neq v, u \neq t, u \neq w\}$

The important result of these extensions is the following:

Proposition 17

Well-typed terms that are irreducible for the system given by $\rightsquigarrow$ and $\succsim$ are without cut.

▷ This result holds due to the fact that this new system allows cuts to be commuted up with all other constructions except itself, and it can also be reduced under all those constructions, that were stopping the reduction in the original reduction system. The proof is then done like above by induction on the sum of the sizes of the immediate subterms of the cut:

- the base cases are the same as above, since$\rightsquigarrow$ has the exact same rules as $\rightarrow$ with only an extended scope of action
- if a clash occurs between a cut and any other rule, they now commute with $\succsim$ and the sum of the size of the immediate subterms is thus reduced by 1
- if a clash occurs between two cuts, then $P = (P_1||_x P_2)||_y P_3$ and, by induction hypothesis, the subterm $P_1||_x P_2$ reduces, so the whole term reduces according to this reduction.

□

This extension does not match how programs execute in the π-calculus model, akin to how general β-reduction does not match the execution of programs in the λ-calculus model. For example it gives an equivalence between terms $u_x(t).\bar{v}_y\langle w\rangle.P$ and $\bar{v}_y\langle w\rangle.u_x(t).P$, even though they do not behave the same in the projection. However, like the general β-reduction, it will allow for cut elimination. Also, the way this extension is done, it is immediate by extending the proofs above, since the cases are all treated the same, that:

Proposition 18

Relation $\rightsquigarrow$ is strongly confluent.

Proposition 19

The reflexive closure of $\succsim$ is a simulation for $\rightsquigarrow$.

Proposition 20

Relation $\succsim$ is confluent up to $\cong$.

Proposition 21

For all $P \cong Q$ and $P \rightsquigarrow P'$,
there exists Q', Q'', P'' such that:

$$\begin{array}{ccc} P & \cong & Q \\ \downarrow & & \wr\Upsilon^* \\ P' & & Q' \\ \wr\Upsilon^* & & \downarrow \\ P'' & \cong & Q'' \end{array}$$

Proposition 22

Relations $\rightsquigarrow$ and $\succsim$ preserve typing.

Proposition 23

The rewriting system induced by $\rightsquigarrow$ and $\succsim$ terminates

Finally, we can prove cut elimination for the extended system:

Corollary 24

The reduction system given by rules $\rightsquigarrow$ and $\succsim$ eliminates cuts (up to $\cong$),
ie. for all $P \vdash \Gamma$ provable in the typing system, there exists P' and a proof of $P \vdash \Gamma$ with $P' \cong P$.

$\triangleright$ It is an immediate corollary of the above propositions and extension, as the reduction terminates, preserves typing and has irreducible terms be without cut. $\square$

Remark: with no exponential, it is no surprise that cut elimination terminates.

Remark: one could also go the way of trying to prove the cut elimination in quantifier-closed context, that is closing the term by cutting successively for all variables of a quantified type against a term with the same variable of an opposite type, so as to end up with a contextualized term which typing does not contain any variable with a quantified type. That lifts the hypothesis of extending the reduction to something that does not match the execution of terms, but the proof also becomes more technical and tedious, and will not be attempted here. There might still be a problem with the other prefixes though, as well as a need to allow for two cuts to commute with each other, that could be added in the congruence $\equiv$.

Remark: this extension does not fit well in the projection. That is normal, as it is not faithful to the way processes execute, and so it does not respect the way execution in the classic model is defined. Especially, two terms that are congruent for $\cong$ might not be for the projection, for example $u_x(v).P||_y Q \cong u_x(v).(P||_y Q)$ but $u(v).P|Q \not\equiv u(v).(P|Q)$.

Example: let us take back the example from part 1, as we now have the tools to eliminate the cuts in it:

$$\begin{aligned}(\nu v)(\bar{u}_x\langle v\rangle.(x \leftrightarrow y)||_y v_y(t').P)||_x u_x(t).(\bar{t}_x\langle w\rangle.(x \leftrightarrow z)||_z Q) &\succ (\nu v)(\bar{u}_x\langle v\rangle.(x \leftrightarrow y)||_y v_y(t').P||_x u_x(t).(\bar{t}_x\langle w\rangle.(x \leftrightarrow z)||_z Q)) \\ &\succsim^2 (\nu v)(\bar{u}_x\langle v\rangle.((x \leftrightarrow y)||_y v_y(t').P)||_x u_x(t).\bar{t}_x\langle w\rangle.((x \leftrightarrow z)||_z Q)) \\ &\rightsquigarrow^2 (\nu v)(\bar{u}_x\langle v\rangle.v_x(t').P[x/y]||_x u_x(t).\bar{t}_x\langle w\rangle.Q[x/z]) \\ &\to (\nu v)(v_x(t').P[x/y]||_x \bar{v}_x\langle w\rangle.Q[x/z]) \\ &\to (\nu v)(P[x/y][w/t']||_x Q[x/z])\end{aligned}$$

and the elimination continues on P and Q.
Since we showed that type was preserved by those operations, the type from above still fits the reduced term:

$$(\nu v)(P[x/y][w/t']||_x Q[x/z]) \vdash \Gamma, \Delta$$

with $P[x/y][w/t'] \vdash \Gamma, x : A[w/t']$ and $Q[x/z] \vdash \Delta, x : A[w/t']^{\perp}$.

Conclusion

During this internship, we worked first on understanding how π-calculus worked, and looked at simple typing systems for them. Then we looked specifically at the example shown in [5] and tried to take that as a base to construct something smaller (no exponential) but that would have a good proof of cut elimination. This is what lead to this work, after many steps of trial and error. This is still not perfect though, as there are many choices made for cut elimination, especially the need to either be able to commute all prefixes with the cut, or the need to be able to commute internal action prefixes with the cut and the cut with itself. There are also many ways to extend this work still, first by extending the system to the additive parts of LL and the π-calculus, and then adapting the methods to be able to work with the exponential constructs (exponential modalities in MELL and replication in the π-calculus) as well.

One of the main limitations of this work is the choice to treat the prefixes that project to τ in the extended system. Although it allows to stay close to the usual calculus, a version with those rules in $\succ$ and $\equiv$ (but with the commutation of action prefixes still left out) would look a lot more like proof nets than this version, and the commutation of λ and ϵ especially would be a reasonable equivalent to the reversibility of $\parr$ and $\bot$ in the logic part. This version would in turn need some work on the π-calculus side to allow for commutations of π with the parallel, and maybe defining an observational preorder on the π-calculus at this time.

There is also a limitation in that $\equiv$ is not a bisimulation for $\rightarrow$, so this needs to be worked on to either find a good relation and a good congruence to have that, or to write just the right theorem for the confluence of diagrams involving a reduction against a congruence (because $\succ^*$ clearly is too much, but finding the minimal amount of $\succ$ needed for it to work needs research). That is a major difference with the classic calculus, which works up to congruence from the beginning.

The last limitation of importance, that is tied to the first one in the first paragraph, is the behavior of $\rightsquigarrow$ and $\gtrsim$ in regards to the projection, as they do not project to usual known reductions and congruences anymore, but one might be able to define an observational preorder to go around this difficulty. This solution might not be ideal though, research is needed on that as well.

Still, the result is promising, with many ways to improve it and trails to follow for the future, and the properties that emerged from this construct are interesting and welcome for a typing and reduction system on processes. An other thing to study would be the possibility to reverse the projection, that is see if, with a working execution in π-calculus, we can deduce an annotation that follows this very execution.

At last, the relation of this system to other existing ones is also important, as there is probably a projection of this system's sequents in other existing systems, perhaps i/o-types with linearity would be a good lead to start.

Thanks

I would like to thank my tutor for this internship, Emmanuel Beffara, first, because he was the one who came up with the subject and he was of great help all along, the discussions we had made the work efficient and fast enough to have that much in only 3 months. I want to thank my teachers as well, especially the logic team here, for my first internship here two years ago was what had me interested in mathematical logic in the first place. At last, thank you to all people who had to bear with me during this internship (and in general), notably my friends here, your support was of great help, and to my best friend whose support is so important even though we live so far away.

References

[1] Davide Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[2] Gianluigi Bellin and Philip J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, December 1994.

[3] Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, March 2016.

[4] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, September 1999.

[5] Emmanuel Beffara. Unifying type systems for mobile processes. April 2015.