PLACEHOLDER

Julien Gabet

Mars-June, 2018

Introduction

Theory of mobile processes is an important domain for research today, be it for its applications on network interactions (be it physical networks like information or energy distribution, railways...), or for its importance in mathematical logic (proof networks, concurrency and continuation transmission are some of the main points we could think of). A usual theory of mobile processes is the π-calculus, on which some work has been done and many others are still ongoing. A reference for its introduction would be *The π-calculus, A Theory of Mobile Processes* by Davide Sangiorgi and David Walker [1]. Especially, the π-calculus as introduced in all generality is strongly non-deterministic in the reduction of its processes. A lot of work is put into circumventing that, most of which involve developing typing systems for the π-calculus that help induce one reduction for a given typed term. Most of those works only type a fragment of the π-calculus though, and there is much work to do to try and unify these typing systems as well.
We will go the route of constructing a typing 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. Preceding works in that direction include *Unifying Type Systems for Mobile Processes* by Emmanuel Beffara [2], which is the base of this work.
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 do that by introducing a decorated form of the calculus, which annotations will allow us to follow interactions between parts of the terms to facilitate typing, by watching over one variable for each interaction. This way, we allow ourselves to embed a proof of (part of) a correct reduction path for the term, without having the difficulty of a non-deterministic behavior.
To make sure this work is well built, we start from classic 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 typing 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 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 will be focused 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 will 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 will introduce the typing system and show results such as type preservation for the reduction system defined before, and a way to eliminate cut by extending it.

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:

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

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.

With this system defined, we would like to look at an example term that would be interesting to follow in the paper.

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 $(\nu v)(\bar{u}(v)|v(t).P)$, where we forget the trailing 0 after the sending part. It could then interact with anything of the form $u(t).(\bar{t}(w)|Q)$, where we again forget the trailing zero after the receiving part.
Putting these together:

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

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 $\rightarrow$ 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.

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:

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

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

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

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: the parallel and synchronization constructions behave differently on their variable. The parallel rule gives a term that puts together the two instances of the variable observed, and keeps it available for further use, while the synchronization rule uses both instances of the variable observed and thus has it unavailable for further use in the created term. The synchronization is a binding operator.

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 basically is that the single one 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 h

EXAMPLE + PROJECTION HERE + PROJECT λ and ϵ to τ.

NEW SECTION 2 HERE

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 3

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

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

We also define rules for commuting the synchronization with other rules, at first not holding under action prefixes:

$$\begin{array}{rlr}
(P|_x Q)||_y R \succ (P||_y R)|_x Q & & \textit{symmetric in } ||,\ y \notin fv(Q) \quad 5a \\
(P|_x Q)||_y R \succ P|_x(Q||_y R) & & \textit{symmetric in } ||,\ y \notin fv(P) \quad 5b \\
\epsilon_x.P||_y Q \succ \epsilon_x.(P||_y Q) & & \textit{symmetric in } ||,\ x \notin fv(Q) \quad 6a \\
\lambda_x y.P||_z Q \succ \lambda_x y.(P||_z Q) & & \textit{symmetric in } ||,\ y \notin fv(Q) \quad 6b \\
(\nu u)P||_x Q \succ (\nu u)(P||_x Q) & & \textit{symmetric in } ||,\ u \notin fn(Q) \quad 6c
\end{array}$$

Remark: In rules 5 and 6, some parts have strong requirements, such as a variable 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, when no other rule can apply. See an example below of a term annotated to be able to reduce, and the same term with annotations changed that fails to do so.

INCLUDE EXAMPLE HERE (and everywhere really...)

We would like these arrows to have a confluent behavior. First, we remark that $\to$ is strongly confluent. Then, we will need an equivalence relation to be defined for the confluence of $\succ$.

Proposition 4

Relation $\to$ is strongly confluent.

$\triangleright$ 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 equalizer base terms synchronized on the same variable, but the commutativity of this construction makes it close immediately:

$$\begin{tikzcd}
& x \leftrightarrow y||_x x \leftrightarrow z \arrow[dl] \arrow[dr] & \\
(x \leftrightarrow z)[y/x] \arrow[d, phantom, "\shortparallel"] & & (x \leftrightarrow y)[z/x] \arrow[d, phantom, "\shortparallel"] \\
y \leftrightarrow z & = & z \leftrightarrow y
\end{tikzcd}$$

□

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

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

Definition 5

- *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 6

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

$\triangleright$ The cases to treat are those where $x \leftrightarrow y$ interacts with something. We treat two of them below, and also put the cut variable in exponent of the term where it appears in, in order to apply rule 5 efficiently, where appilcable:

$$\begin{tikzcd}
(P^y|_xQ)||_y x \leftrightarrow y \arrow[r] \arrow[d, "\succ_{5a}"] & (P^y|_xQ)[x/y] = P[x/y]|_xQ \\
(P^y||_y x \leftrightarrow y)|_xQ \arrow[ur] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\lambda_x y.P||_z z \leftrightarrow x \arrow[r] \arrow[d, "\succ_{6b}"] & (\lambda_x y.P)[x/z] = P[x/z] \\
\lambda_x y.(P||_z z \leftrightarrow x) \arrow[ur] &
\end{tikzcd}$$

All other cases act the same way, or are the same 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. A congruence is needed to make this relation confluent.

Definition 7

We define a congruence rule, that does not act on action prefixes, with the following axioms:

$$\begin{array}{rcll}
(P|_xQ)|_yR & \equiv & (P|_yR)|_xQ & y \notin fv(Q), x \notin fv(R) \quad a_1 \\
P|_x(Q|_yR) & \equiv & Q|_y(P|_xR) & y \notin fv(P), x \notin fv(Q) \quad a_2 \\
(P|_xQ)|_yR & \equiv & P|_x(Q|_yR) & y \notin fv(P), x \notin fv(R) \quad b \\
P|_x\alpha_y.Q & \equiv & \alpha_y.(P|_xQ) & \textit{symmetric in } |, \alpha. \in \{\epsilon., \lambda.z\}, y, t \notin fv(P) \quad c \\
\alpha_x.\beta_y.P & \equiv & \beta_y.\alpha_x.P & \alpha., \beta. \in \{\epsilon., \lambda.z\}, x \neq y \quad d \\
(\nu u)P|_xQ & \equiv & (\nu u)(P|_xQ) & \textit{symmetric in } |, u \notin fn(Q) \quad e \\
(\nu u)\alpha_x.P & \equiv & \alpha_x.(\nu u)P & \alpha. \in \{\epsilon., \lambda.z\} \quad f
\end{array}$$

This rule is defined as an equivalence (reflexive, symmetric and transitive), and applies inside contexts (except action prefixes).

This equivalence allows for the confluence of $\succ$:

Proposition 8

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

$$\begin{tikzcd}
& P \arrow[dl, "\succ"'] \arrow[dr, "\succ"] & \\
P_1 \arrow[d, "\succ"] & & P_2 \arrow[d, "\succ"] \\
P_1' & \equiv & P_2'
\end{tikzcd}$$

▷ 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):

$$\begin{tikzcd}
 & (P^y|_xQ)||_y(R^y|_zS) \arrow[dl, "{5a,\ left}"'] \arrow[dr, "{5a,\ right}"] & \\
(P^y||_y(R^y|_zS))|_xQ \arrow[d, "5a"] & & ((P^y|_xQ)||_yR^y)|_zS \arrow[d, "5a"] \\
((P^y||_yR^y)|_zS)|_xQ & \equiv_{a_1} & ((P^y||_yR^y)|_xQ)|_zS
\end{tikzcd}$$

$$\begin{tikzcd}
 & (P^y|_xQ)||_y(R|_zS^y) \arrow[dl, "{5a,\ left}"'] \arrow[dr, "{5b,\ right}"] & \\
(P^y||_y(R|_zS^S))|_xQ \arrow[d, "5b"] & & R|_z((P^y|_xQ)||_yS^y) \arrow[d, "5a"] \\
(R|_z(P^y||_yS^y))|_xQ & \equiv_b & R|_z((P^y||_yS^y)|_xQ)
\end{tikzcd}$$

Rule 6 against rule 6 works the same, here is an example with ϵ and ν:

$$\begin{tikzcd}
 & \epsilon_x.P||_y(\nu u)Q \arrow[dl, "6a"'] \arrow[dr, "6c"] & \\
\epsilon_x.(P||_y(\nu u)Q) \arrow[d, "6c"] & & (\nu u)(\epsilon_x.P||_yQ) \arrow[d, "6a"] \\
\epsilon_x.(\nu u)(P||_yQ) & \equiv_f & (\nu u)\epsilon_x.(P||_yQ)
\end{tikzcd}$$

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

$$\begin{tikzcd}
 & (P^y|_xQ)||_y(\nu u)R \arrow[dl, "5a"'] \arrow[dr, "6c"] & \\
(P^y||_y(\nu u)R)|_xQ \arrow[d, "6c"] & & (\nu u)((P^y|_xQ)||_yR) \arrow[d, "5a"] \\
(\nu u)(P^y||_yR)|_xQ & \equiv_e & (\nu u)((P^y||_yR)|_xQ)
\end{tikzcd}$$

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

We need $\equiv$ to not alter the behavior of $\rightarrow$ in order for the system to work well. We do not have that kind of property here (*ie.* $\equiv$ is not a simulation for $\rightarrow$), but we can observe an interesting phenomenon by allowing to do $\succ$ steps where needed:

$$\begin{tikzcd}
\lambda_x y.\epsilon_z.P||_x(Q|_xR) \arrow[d] & \equiv & \epsilon_z.\lambda_x y.P||_x(Q|_xR) \arrow[d, "\succ"] \\
(\epsilon_z.P||_xQ)||_yR[y/x] \arrow[d, "\succ"] & & \epsilon_z.(\lambda_x y.P||_x(Q|_xR)) \arrow[d] \\
(\epsilon_z.(P||_xQ))||_yR[y/x] & \succ & \epsilon_z.((P||_xQ))||_yR[y/x])
\end{tikzcd}$$

We have a more general property emanating of that, that gives us something close to a simulation behavior:

Proposition 9

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

ie.
$$\begin{tikzcd}
P \arrow[d] & \equiv & Q \arrow[d, "\succ^*"] \\
P' \arrow[d, "\succ^*"] & & Q' \arrow[d] \\
P'' & \equiv & Q''
\end{tikzcd}$$

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

$$\begin{tikzcd}
\epsilon_z.\lambda_x y.P||_z0_z \arrow[d] & \equiv & \lambda_x y.\epsilon_z.P||_z0_z \arrow[d, "\succ"] \\
\lambda_x y.P & & \lambda_x y.(\epsilon_z.P||_z0_z) \arrow[ll]
\end{tikzcd}$$

$$\begin{tikzcd}
\epsilon_z.((P|_xQ)|_yR)||_z0_z \arrow[d] & \equiv & \epsilon_z.(P|_x(Q|_yR))||_z0_z \arrow[d] \\
(P|_xQ)|_yR & \equiv & P|_x(Q|_yR)
\end{tikzcd}$$

$$\begin{tikzcd}[row sep=small]
((P|_x Q^y)|_y R^y)||_y \lambda_y z.S \arrow[d] \arrow[r, phantom, "\equiv"] & (P|_x(Q^y|_y R^y))||_y \lambda_y z.S^y \arrow[d, phantom, sloped, "\succ"] \\
(P|_x Q^y)||_y (R[z/y]||_z S) & P|_x((Q^y|_y R^y))||_y \lambda_y z.S^y) \arrow[d] \\
 & P|_x(Q^y||_y (R[z/y]||_z S^y)) \arrow[ul, phantom, sloped, "\prec"]
\end{tikzcd}$$

where $P \equiv P'$:

$$\begin{tikzcd}[row sep=small]
u_x(t).P||_x \bar{u}_x\langle v\rangle.Q \arrow[d] \arrow[r, phantom, "\equiv"] & u_x(t).P'||_x \bar{u}_x\langle v\rangle.Q \arrow[d] \\
P[v/t]||_x Q \arrow[r, phantom, "\equiv"] & P'[v/t]||_x Q
\end{tikzcd}$$

$\square$

2 Typing decorated terms with MLL

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

Definition 10

The language of formulas is given by the following grammar:

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

It is based on the structure of MLL, with atoms formed by the names traded by π-terms, 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 11

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:

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

Construction rules:

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

Action rules:

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

Nu rule:

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

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 $\rightarrow$ and $\succ$ defined in 3. 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.

Proposition 12

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

▷ Show the reduction steps. □

Proposition 13

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

▷ Take all cases in the proof above, and show the strictly decreasing tuple of sizes of terms under the cut. Since $\mathbb{N}$ has no infinite decreasing sequence, by well formed induction, the reduction terminates. □

Remark: This system preserves typing and terminates, but does not completely eliminate cut. For example, terms like $u_x(t).\epsilon_y.P||_y 0_y$ cannot reduce, yet the cut on y is 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:

Definition 14

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

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

$$u_x(t).P||_y Q \succsim u_x(t).(P||_y Q) \qquad \textit{symmetric in } || \quad 6d$$
$$\bar{u}_x\langle v\rangle.P||_y Q \succsim \bar{u}_x\langle v\rangle.(P||_y Q) \qquad \textit{symmetric in } || \quad 6e$$

We finally extend $\equiv$ into $\cong$ by allowing α, β to be action prefixes in rules c, d:

$$\alpha., \beta. \in \{\epsilon., \lambda.z, u.(t), \bar{u}.\langle v\rangle\};$$

as well as in rule f:

$$\alpha. \in \{\epsilon., \lambda.z, v.(t), \bar{v}.\langle w\rangle, u \neq v, u \neq t, u \neq w\}.$$

The important result of these extensions is the following:

Proposition 15

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

$\triangleright$ This result holds 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, *including* action prefixes 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 $\to$ 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 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.

$\square$

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 16

Relation $\rightsquigarrow$ is strongly confluent.

Proposition 17

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

Proposition 18

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

Proposition 19

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

$$\begin{tikzcd}
P \arrow[r, phantom, "\cong"] \arrow[d, rightsquigarrow] & Q \arrow[d, "\succsim^*"] \\
P' \arrow[d, "\succsim^*"] & Q' \arrow[d, rightsquigarrow] \\
P'' \arrow[r, phantom, "\cong"] & Q''
\end{tikzcd}$$

Proposition 20

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

Proposition 21

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

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

Corollary 22

The reduction system given by rules $\rightsquigarrow$ and $\precsim$ 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.

Also, insert here the followed example from the other paper that is to be shown in part 1, to show here that it is not only reducing well, but also well-typed. (Question for later: Is there a term, simplest possible if exists, that reduces well without typing, but cannot be well-typed?)

3 Links with the usual π-calculus

Definition 23

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

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

We also define a structural congruence on terms as follows:

$$\begin{aligned} P|0 &\equiv P \\ P|Q &\equiv Q|P \\ P|(Q|R) &\equiv (P|Q)|R \\ (\nu u)P &\equiv P \quad \text{if } u \text{ is not a free name in } P \end{aligned}$$

and a reduction rule:

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

We remark most of the important constructions we use in the decorated calculus are here as well, especially being able to specify a name being private, running two processes in parallel as well as send or receive names over a channel. In fact, the synchronization construction from our annotated calculus is a form of parallelization as well, and in a world with no variables to describe the behavior of terms it makes sense to not have a separate construction for that. Also, the $\leftrightarrow$, ϵ and λ constructions in our annotated system don't do much in the calculus part, aside from allowing to eliminate the synchronization/cut, and their content only affects the typing system. Thus, it makes no sense to have such constructions in a calculus-only system where there is no equivalent to the cut rule from logic systems. We then have covered all constructions from both calculi here, so an interesting thing to do would be to project the annotated calculus that has more constructions in he usual π-calculus and see it we can get some interesting properties out of that.

Definition 24

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

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

The important part to verify is that the projection behaves well with the reduction system proposed in part 1:

Proposition 25

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

$\triangleright$ Do all meaningful constructions here.
In essence: the reduction of $u_x.P||_x\bar{u}_x.Q$ projects to a reduction, everything else projects to structural congruence. Maybe treating this and a few of the structural congruence projections is good, and say that the rest is treated the same way (many similar cases and not really complicated to see, writing everything feels like trying to occupy space just to increase the page count). $\square$

Open about $\rightsquigarrow$ here or in the conclusion, because it does not act like a reduction in the projection, but gives some sort of pre-order on terms.

Conclusion

A nice conclusion and working trails for later here.

- what we did so far: working small part of the calculus, good properties and behavior, reasonable projection
- ideas for the behavior of $\equiv$ as a "bisimulation" in the annotated calculus
- behavior of $\rightsquigarrow$ and $\gtrsim$ with the projection: in the general case you do not want to have an action prefix be able to commute like that, but there is at least an "order" on the capabilities of terms u.P|Q and u.(P|Q)
- next step: work on exponential rules, and additive rules too
- talk about failed attempts here maybe, and/or about misleading intuitions that were explored in the past 2.5 months ?
- more things maybe, if something interesting comes up

References

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

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