Classical Proofs as Parallel Programs*

Federico Aschieri
TU Wien, Vienna, Austria

Agata Ciabattoni
TU Wien, Vienna, Austria

Francesco A. Genco
TU Wien, Vienna, Austria

We introduce a first proofs-as-parallel-programs correspondence for classical logic. We define a parallel and more powerful extension of the simply typed λ-calculus corresponding to an analytic natural deduction based on the excluded middle law. The resulting functional language features a natural higher-order communication mechanism between processes, which also supports broadcasting. The normalization procedure makes use of reductions that implement novel techniques for handling and transmitting process closures.

1 Introduction

The λ-calculus is the heart of functional programming languages. The deep connection between its programs and intuitionistic proofs is known as Curry–Howard correspondence; useful consequences of this correspondence are the termination of well-typed functional programs and the possibility of writing provably correct programs, see, e.g., [24].

The extension of the Curry–Howard correspondence to classical logic came many years later, with Griffin's discovery [10] that Pierce's law $((A \to B) \to A) \to A$ provides a type for the call/cc operator of Scheme. Since then many λ-calculi motivated by the correspondence with classical logic have been introduced. Remarkably, different formalizations of the same logic lead to different results. In particular computational interpretations of classical logic are very sensible to the selected logical formalism and the concrete coding of classical reasoning. The main two choices for the former are natural deduction and sequent calculus, while for the latter are Pierce's Law, reasoning by contradiction $(\neg A \to \bot) \to A$, multi-conclusion deduction and the excluded middle law EM $\neg A \vee A$.

For instance, reasoning by contradiction gives rise to control operators and corresponds very directly to Parigot $\lambda\mu$-calculus [17], which relates to classical natural deduction as the λ-calculus relates to intuitionistic natural deduction **NJ** [20]. Examples of multi-conclusion deductions in classical sequent calculus are [4, 22]. These Curry–Howard correspondences match perfectly classical logic and computation: each step of program reduction corresponds to a proof transformation, and the evaluation of λ-terms corresponds to normalization, a procedure that makes proofs *analytic*, i.e. only containing formulas that are subformulas of premises and conclusion.

All the functional programming languages resulting from Curry-Howard correspondences for propositional classical logic model *sequential computation* and extend the λ-calculus via programming concepts such as continuations [15]. Remarkably, none of them is based on $\neg A \vee A$, which appears to be related to some form of *parallelism*, see [5]. A natural way to make this parallelism explicit would be to extend the Curry–Howard correspondence to **NJ** augmented with suitable rules for EM. So far, however, it has been a long-standing open problem to provide an analytic natural deduction based on EM *and* enjoying a significant computational interpretation. In the only known Curry–Howard correspondence for EM-based propositional natural deduction [11] proof-terms are not interpreted as parallel programs and do not correspond to analytic proofs. The calculus in [11] extends the simply typed λ-calculus with an operator

*Supported by FWF: grant Y544-N2 and project W1255-N23.

Submitted to:
Gandalf 2018

for exception handling. The lack of analiticity, however, implies that important reduction rules are missing: there are some exceptions that should be raised, but are not. These exceptions contain free variables which are locally bound and cannot be delivered to the exception handler, otherwise they would become free variables in the whole term. Hence a crucial missing element in [11] is *code mobility*, along with the techniques handling the bindings between a piece of code and its environment [8]. Parallelism and code mobility were instead employed to define the calculus λ_{G} [1] and provide a Curry–Howard correspondence for Gödel logic, a well known logic intermediate between classical and intuitionistic logic.

We exploit here the techniques developed for λ_{G} in order to extract from propositional classical logic a new parallel λ-calculus with a remarkably simpler communication mechanism. The calculus λ_{CL}, as we call it, extends simply typed λ-calculus by a communication mechanism that interprets the natural deduction rule for $A \vee \neg A$. Processes of λ_{CL} only communicate through private channels similar to those bound by the restriction operator ν in the π-calculus, the most widespread formalism for modeling concurrent systems [16, 19]. These channels are introduced by the typing rule for EM and their behavior during communication is defined by λ_{CL} reduction rules. The basic communication reductions, called *basic cross reductions*, behave as follows: let $\mathscr{C}$ be a process ready to send a message to another process $\mathscr{D}$ through a channel a, in symbols $\mathscr{C}[\overline{a}\, u] \parallel_a \mathscr{D}$; then, if u is data or a closed process, the result of the communication is $\mathscr{D}[u/a]$, similarly to asynchronous π-calculus [12] or the concurrent λ-calculus proposed in [3]. Although simple, this mechanism makes λ_{CL} more powerful than simply typed λ-calculus and propositional λ_μ (see Prop. 6). Remarkably it can also model *races*, situations in which several processes compete for a limited amount of resources; moreover, in contrast with calculi based on point-to-point communication [6] as π-calculus or λ_{G} [1] (see Sec. 4) λ_{CL} also renders *broadcasting*.

The normalization of λ_{CL} ensures that natural deduction derivations can be transformed into analytic ones. Similarly to the normalization for λ_{G}, this procedure requires additional reduction rules (*cross reductions*) that offer a solution to a "*fundamental problem in any distributed implementation of a statically-typed, higher-order programming language*": how to send open processes and transmit function closures from one node to another (cf. the description of Cloud Haskell in [7]). Indeed, the process u to be sent might not be closed and might need resources that are or will be available in $\mathscr{C}$. *Cross reductions* allow u to be transmitted and create a new communication channel b for transferring the complete closure of u afterwards. Not even higher-order π-calculus directly supports such a mechanism; let alone π-calculus, whose communication only applies to data or channels. As discussed in Ex. 2, our code mobility can be used for program optimization. The technically sophisticated normalization proof in Sec. 3 adapts the proof for λ_{G} [1].

2 λ_{CL}: a Curry–Howard interpretation of Classical Logic

We introduce λ_{CL}, our typed parallel λ-calculus for Classical Logic. λ_{CL} extends the standard Curry–Howard correspondence [13, 24] for the intuitionistic natural deduction **NJ** [18] by a parallel operator that interprets the rule (EM) for $A \vee \neg A$. Table 1 defines a type assignment for λ_{CL}-terms, called **proof terms** and denoted by $t, u, v \ldots$, which is isomorphic to the natural deduction system $\mathbf{NJ} + (\mathrm{EM})$. The typing rules for axioms, implication, conjunction, disjunction and ex-falso-quodlibet are those for the simply typed λ-calculus [9]. Parallelism is introduced by the (EM) rule . The contraction rule, useful to define parallel terms that do not communicate, is in fact redundant from the point of view of proof theory.

The reduction rules of λ_{CL} are in Figure 1. They consist of the usual simply typed λ-calculus reductions, instances of $\vee$ permutations adapted to the $\parallel$ operator, and new communication reductions: *Basic Cross Reductions* and *Cross Reductions*. Since we are dealing with a Curry–Howard correspondence,

$$x^A : A \qquad \dfrac{u:A \quad v:A}{u \parallel v : A}\ (\text{contr.}) \qquad \dfrac{u:A \quad t:B}{\langle u,t\rangle : A\wedge B} \qquad \dfrac{u:A\wedge B}{u\,\pi_0 : A} \qquad \dfrac{u:A\wedge B}{u\,\pi_1 : B}$$

$$\dfrac{\begin{array}{c}[x^A:A]\\ \vdots \\ u:B\end{array}}{\lambda x^A u : A\to B} \qquad \dfrac{t:A\to B \quad u:A}{tu:B} \qquad \dfrac{\begin{array}{cc}[a^{\neg A}:\neg A] & [a^A:A]\\ \vdots & \vdots\\ u:B & v:B\end{array}}{u \parallel_a v : B}\ (\text{EM}) \qquad \begin{array}{c}\dfrac{u:\bot}{u\,\mathsf{efq}_P : P}\\ \text{with } P \text{ atomic and } P\neq\bot\end{array}$$

where all the occurrences of a in u and v are respectively of the form $a^{\neg A}$ and a^A

Table 1: Type assignments for λ_{CL} terms and natural deduction rules.

every reduction rule of λ_{CL} corresponds to a reduction for the natural deduction calculus $\mathbf{NJ}+(\mathrm{EM})$.

Before explaining the calculus reductions, we recall the essential terminology and definitions, see e.g. [9]. Proof terms may contain variables $x_0^A, x_1^A, x_2^A, \ldots$ of type A for every formula A; these variables are denoted by x^A, y^A, $z^A, \ldots, a^A, b^A, c^A$ and whenever the type is irrelevant by $x,y,z,\ldots,a,b$. For clarity, the (EM) rule variables will be often denoted by $a,b,c,\ldots$ but they are not in a syntactic category apart. A variable x^A that occurs in a term of the form $\lambda x^A u$ is called **λ-variable** and a variable a that occurs in a term $u \parallel_a v$ is called **channel** or **communication variable** and represents a *private* communication channel between the processes u and v. We adopt the convention that $a^{\neg A}$ and a^A are denoted by $\overline{a}$ and a respectively, where unambiguous. Free and bound variables of a term are defined as usual. For the new term $u \parallel_a v$, all free occurrences of a in u and v are bound in $u \parallel_a v$. We assume the standard renaming rules and α-equivalences that are used to avoid the capture of variables in the reductions.

Notation. The connective $\to$ and $\wedge$ associate to the right and by $\langle t_1,t_2,\ldots,t_n\rangle$ we denote the term $\langle t_1,\langle t_2,\ldots\langle t_{n-1},t_n\rangle\ldots\rangle\rangle$ (which is $\mathtt{tt} : \top$ if $n=0$) and by π_i, for $i=0,\ldots,n$, the sequence $\pi_1\ldots\pi_1\pi_0$ selecting the $(i+1)$th element of the sequence. The expression $A_1\wedge\ldots\wedge A_n$ denotes $\top$ if $n=0$. As usual, we use $\neg A$ as shorthand notation for $A\to\bot$.

We write $\Gamma\vdash t:A$ if $\Gamma = x_1:A_1,\ldots,x_n:A_n$ and all free variables of a proof term $t:A$ are in $x_1,\ldots,x_n$. From the logical point of view, t represents a natural deduction of A from the hypotheses $A_1,\ldots,A_n$. If the symbol $\parallel$ does not occur in t, then t is a **simply typed λ-term** representing an intuitionistic deduction.

We first explain the cross reductions from a proof-theoretical point of view. Basic cross reductions correspond to the following transformation of natural deduction derivations

$$\dfrac{\begin{array}{cc}\dfrac{[\neg A] \quad \begin{array}{c}\delta\\ A\end{array}}{\bot} & [A]\\ \vdots & \vdots\\ C & C\end{array}}{C} \quad\mapsto\quad \begin{array}{c}\delta\\ A\\ \vdots\\ C\end{array}$$

where no assumption in δ is discharged below $\bot$ and above C. When this is the case, intuitively, the displayed instance of (EM) is hiding some redex that should be reduced. The reduction precisely exposes this potential redex [18] and we are thus able to reduce it. More instances of $\neg A$ and A might occur in the respective branches. This, in combination with the contraction rule (see Table 1), gives rise to races and broadcasting, as explained in the computational interpretation below. Cross reductions correspond to

$$\dfrac{\begin{array}{cc}\dfrac{[\neg A]^1 \quad \begin{array}{c}[\Gamma]\\ \delta\\ A\end{array}}{\bot} & [A]^1\\ \vdots & \vdots\\ C & C\end{array}}{C} \quad\mapsto\quad \dfrac{\dfrac{\begin{array}{cc}\dfrac{[\neg\bigwedge\Gamma]^2 \quad \dfrac{[\Gamma]}{\bigwedge\Gamma}}{\bot} & [A]^1\\ \vdots & \vdots\\ C & C\end{array}}{C}\,{}_1 \quad \begin{array}{c}\dfrac{[\bigwedge\Gamma]^2}{\Gamma}\\ \delta\\ A\\ \vdots\\ C\end{array}}{C}\,{}_2$$

As before, in the right derivation we prove by δ all assumptions A, but now we also need to discharge the assumptions Γ of δ in the rightmost branch; which are discharged in the left derivation between $\bot$ and C. This is done by 2: a new application of (EM) to the conjunction $\bigwedge\Gamma$ of such assumptions. Accordingly, we use $\dfrac{\neg\bigwedge\Gamma \qquad \bigwedge\Gamma}{\bot}$ in the leftmost branch. To discharge the remaining occurrences of $\neg A$, we need to keep the original instance 1 of (EM), and thus the central branch of the resulting proof is just a duplicate.

Before discussing the computational content of the calculus we introduce a few more definitions.

Definition 1 (Simple Parallel Term)**.** A **simple parallel term** is a λ_{CL}-term $t_1 \parallel \ldots \parallel t_n$, where each t_i, for $1 \leq i \leq n$, is a simply typed λ-term.

Definition 2. A **context** $\mathscr{C}[\,]$ is a λ_{CL}-term with some fixed variable $[\,]$ occurring exactly once.

- A **simple context** is a context which is a simply typed λ-term.
- A **simple parallel context** is a context which is a simple parallel term.

For any λ_{CL}-term u with the same type as $[\,]$, $\mathscr{C}[u]$ denotes the term obtained replacing $[\,]$ with u in $\mathscr{C}[\,]$, *without renaming bound variables*.

Definition 3 (Multiple Substitution)**.** Let u be a proof term, $x = x_0^{A_0}, \ldots, x_n^{A_n}$ a sequence of variables and $v : A_0 \wedge \ldots \wedge A_n$. The substitution $u^{v/x} := u[v\pi_0/x_0^{A_0} \ldots v\pi_n/x_n^{A_n}]$ replaces each variable $x_i^{A_i}$ of any term u with the ith projection of v.

Basic Cross Reductions can be fired whenever the free variables of t are also free in $\mathscr{C}[\overline{a}\,t]$. In particular, t may represent executable code or data and directly replaces all occurrences of the channel endpoint a. In case there is only one sender and one receiver, the reduction

$$\mathscr{C}[\overline{a}\,t] \parallel_a \mathscr{D} \;\mapsto\; \mathscr{D}[t/a]$$

corresponds to the reduction axiom of the asynchronous π-calculus [12]. In general, $\mathscr{C}[\overline{a}\,t]$ has the shape

$$\mathscr{C}_1 \parallel \ldots \parallel \mathscr{C}_i[\overline{a}\,t] \parallel \ldots \parallel \mathscr{C}_n$$

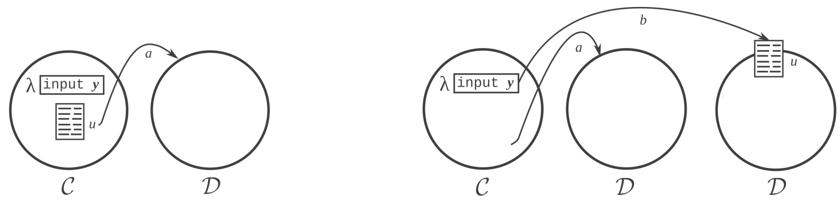
where more than one process might have a message to send. In this case, there is a *race* among the processes $\mathscr{C}_j$ that contain some message $\overline{a}\,t_j$ and compete to transmit it to $\mathscr{D}$. The sender $\mathscr{C}_i[\overline{a}\,t]$ is selected non-deterministically and communicates its message to $\mathscr{D}$:

$$(\mathscr{C}_1 \parallel \ldots \parallel \mathscr{C}_i[\overline{a}\,t] \parallel \ldots \parallel \mathscr{C}_n) \parallel_a \mathscr{D} \;\mapsto\; \mathscr{D}[t/a]$$

Since the receiving term $\mathscr{D}$ exhausts all its channels $a : A$ to receive t, we remove all processes containing $\overline{a} : \neg A$ and obtain a term without a. We also point out that $\mathscr{D}$ is an arbitrary term, so it may well be a sequence of parallel process $\mathscr{D}_1 \parallel \cdots \parallel \mathscr{D}_m$. In this case, $\mathscr{C}_i[\overline{a}\,t]$ *broadcasts* its message t to $\mathscr{D}_1, \ldots, \mathscr{D}_m$:

$$\mathscr{D}[t/a] = \mathscr{D}_1[t/a] \parallel \cdots \parallel \mathscr{D}_m[t/a]$$

Cross Reductions address a crucial problem of functional languages with higher-order message exchange: transmitting function closures (see [7]). The solution provided by our logical types is that function closures are transmitted in two steps: first, the function code, then, when it is available, the evaluation environment. As a result, we can communicate open λ_{CL}-terms which are closed in their original environment, and fill later their free variables, when they will be instantiated. The process of handling and transmitting function closures is typed by a new instance of (EM). For example: assume that a channel a is used to send an arbitrary sub-process u from a process $\mathscr{C}$ to a process $\mathscr{D}$ (below left). Since u might not be closed, it might depend on its environment for providing values for its free variables – y in the example is bound by a λ outside u. This issue is solved in the cross reduction by a fresh channel b which redirects y – the remaining part of the closure – to the new location of u (below right).



The old channel a is kept for further messages that $\mathscr{C}$ might want to exchange with $\mathscr{D}$. Technically, the cross reduction has this shape: $(\ldots \parallel \mathscr{C}[\overline{a}\, u] \parallel \ldots) \parallel_a \mathscr{D} \mapsto (\mathscr{C}[\overline{b}\, \langle y\rangle] \parallel_a \mathscr{D}) \parallel_b \mathscr{D}[u^{b/y}/a]$ On the one hand, the open term u is replaced in $\mathscr{C}$ by the new channel $\overline{b}$ applied to the sequence y of the free variables of u; on the other hand, u is sent to the term $\mathscr{D}$ as $u^{b/y}$, so its free variables are removed and replaced by the channel endpoint b that will receive their future instantiation.

Communication Permutations The only permutations for $\parallel$ that are not standard $\vee$-permutation-like are

$$(u \parallel_a v) \parallel_b w \mapsto (u \parallel_b w) \parallel_a (v \parallel_b w) \quad \text{and} \quad w \parallel_b (u \parallel_a v) \mapsto (w \parallel_b u) \parallel_a (w \parallel_b v)$$

These kind of permutations are between parallel operators themselves and address the *scope extrusion* issue of private channels. For instance, let us consider the term $(v \parallel_a \mathscr{C}[b\,a]) \parallel_b w$ Here the process $\mathscr{C}[b\,a]$ wishes to send the channel a to w along the channel b, but this is not possible being the channel a private. This issue is solved in the π-calculus using the congruence $\nu a(P\,|\,Q)\,|\,R \equiv \nu a(P\,|\,Q\,|\,R)$. Classical logic offers and actually forces a different solution, which is not just permuting w inward but also duplicating it:

$$(v \parallel_a \mathscr{C}[b\,a]) \parallel_b w \mapsto (v \parallel_b w) \parallel_a (\mathscr{C}[b\,a] \parallel_b w) \quad \text{after this reduction } \mathscr{C}[b\,a] \text{ can send } a \text{ to } w.$$

We provide now the last definitions needed to formally define the reduction rules of λ_{CL}. We start with the notion of *strong subformula*, which is key for proving Normalization (Section 3).

Definition 4 (Prime Formulas and Factors [14])**.** A formula is said to be **prime** if it is not a conjunction. Every formula is a conjunction of prime formulas, called **prime factors**.

Definition 5 (Strong Subformula [1])**.** B is said to be a **strong subformula** of a formula A, if B is a proper subformula of some prime proper subformula of A.

Note that here prime formulas are either atomic formulas or arrow formulas, so a strong subformula of A must be actually a proper subformula of an arrow proper subformula of A. The following characterization from [1] of the strong subformula relation will be often used.

Proposition 1 (Characterization of Strong Subformulas)**.** *If B is a strong subformula of A:*

- *if $A = A_1 \wedge \ldots \wedge A_n$, $n > 0$ and $A_1, \ldots, A_n$ are prime, then B is a proper subformula of some $A_1, \ldots, A_n$;*
- *if $A = C \to D$, then B is a proper subformula of a prime factor of C or D.*

Unrestricted cross reductions do not always terminate. Consider, for example, the following loop

$$\lambda y^B\, a^{\neg B}\, y \parallel_a x^{B\to\neg B}\, a^B \;\mapsto\; (\lambda y\, b\, y \parallel_a x\, a) \parallel_b x\, b \;\mapsto\; \lambda y\, b\, y \parallel_b x\, b \tag{1}$$

Intuitionistic Reductions

$$(\lambda x^A\, u)t \mapsto u[t/x^A] \qquad \langle u_0, u_1\rangle\, \pi_i \mapsto u_i, \text{ for } i = 0,1$$

Parallel Operator Permutations

$w(u \parallel_a v) \mapsto wu \parallel_a wv$ if a does not occur free in w

$(u \parallel_a v)\xi \mapsto u\xi \parallel_a v\xi$ if ξ is a one-element stack and a does not occur free in ξ

$w(u \parallel v) \mapsto wu \parallel wv \quad (u \parallel v)\xi \mapsto u\xi \parallel v\xi \quad \lambda x^A\,(u \parallel_a v) \mapsto \lambda x^A\, u \parallel_a \lambda x^A\, v \quad \langle u \parallel_a v, w\rangle \mapsto \langle u, w\rangle \parallel_a \langle v, w\rangle$

$\langle w, u \parallel_a v\rangle \mapsto \langle w, u\rangle \parallel_a \langle w, v\rangle \; \lambda x^A\,(u \parallel v) \mapsto \lambda x^A\, u \parallel \lambda x^A\, v \quad \langle u \parallel v, w\rangle \mapsto \langle u, w\rangle \parallel \langle v, w\rangle \quad \langle w, u \parallel v\rangle \mapsto \langle w, u\rangle \parallel \langle w, v\rangle$

$(u \parallel_a v) \parallel_b w \mapsto (u \parallel_b w) \parallel_a (v \parallel_b w)$ if the communication complexity of b is greater than 0

$w \parallel_b (u \parallel_a v) \mapsto (w \parallel_b u) \parallel_a (w \parallel_b v)$ if the communication complexity of b is greater than 0

$(u \parallel v) \parallel_b w \mapsto (u \parallel_b w) \parallel (v \parallel_b w)$ if the communication complexity of b is greater than 0

$w \parallel_b (u \parallel v) \mapsto (w \parallel_b u) \parallel (w \parallel_b v)$ if the communication complexity of b is greater than 0

Communication Reductions

Basic Cross Reductions $\qquad \mathscr{C}[\overline{a}\, u] \parallel_a \mathscr{D} \;\mapsto\; \mathscr{D}[u/a]$

where $\overline{a} : \neg A, a : A, \mathscr{C}[\,]$ is a context; the free variables of u are also free in $\mathscr{C}[\overline{a}\, u]$; $\overline{a}$ does not occur in u ; and the communication complexity of a is greater than 0.

Cross Reductions $\qquad u \parallel_a v \mapsto u$ if a does not occur in u and $u \parallel_a v \mapsto v$ if a does not occur in v

$$(\ldots \parallel \mathscr{C}[\overline{a}\, u] \parallel \ldots) \parallel_a \mathscr{D} \mapsto \; (\mathscr{C}[\overline{b}\, \langle y\rangle] \parallel_a \mathscr{D}) \; \parallel_b \; \mathscr{D}[u^{b/y}/a]$$

where $\overline{a} : \neg A, a : A$; $(\ldots \parallel \mathscr{C}[\overline{a}\, u] \parallel \ldots)$ is a normal simple parallel term; y is the non-empty sequence of free variables of u bound in $\mathscr{C}[\overline{a}\, u]$; B is the conjunction of the types of the variables in y and b/y is a multiple substitution of these variables; a is rightmost in $\mathscr{C}[\overline{a}\, u]$; b is fresh; $\overline{b} : \neg B, b : B$; the communication complexity of a is greater than 0.

Figure 1: Reduction Rules for λ_{CL}

To avoid such situations we need conditions on the application of cross reductions. As shown below, our conditions are based on the complexity of the channel a of a term $u \parallel_a v$, and are determined using logic. We consider the type B such that a occurs with type $\neg B$ in u and thus with type B in v, the type A of the term $u \parallel_a v$, and the types of its free variables $x_1^{A_1}, \ldots, x_n^{A_n}$. The Subformula Property tells us that, no matter what our notion of computation will turn out to be, when the computation is done, no object whose type is more complex than the types of the inputs and the output should appear. If the prime factors of the types B are not subformulas of $A_1, \ldots, A_n, A$, then these prime factors should be taken into account in the complexity measure we are looking for. This leads to the following definition.

Definition 6 (Communication Complexity)**.** Let $u \parallel_a v : A$ a proof term with free variables $x_1^{A_1}, \ldots, x_n^{A_n}$. Assume that $\overline{a} : \neg B$ occurs in u and $a : B$ occurs in v.

- B is the **communication kind** of a.
- The **communication complexity** of a is the maximum among 0 and the number of symbols of the prime factors of B that are neither proper subformulas of A nor strong subformulas of any $A_1, \ldots, A_n$.

To fire a cross reduction for $u \parallel_a v$ we require that the communication complexity of a is greater than 0. As this is a warning that the Subformula Property does not hold, we are using a logical property as a *computational criterion* for determining when a computation should start and stop. To see it at work, consider again the term $\lambda y^B\, a^{\neg B}\, y \parallel_a x^{B \to \neg B}\, a^B : \neg B$ in the reduction (1). Since all prime factors of the communication kind B of a are proper subformulas of the type $\neg B$ of the term, the communication complexity of a is 0 and the cross reduction is not fired, thus avoiding the loop in (1).

Finally, we recall the notion of stack [15]: a series of operations and arguments.

Definition 7 (Stack)**.** A **stack** is a, possibly empty, sequence $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$ such that for every $1 \le i \le n$, exactly one of the following holds: $\sigma_i = t$, with t proof term or $\sigma_i = \pi_j$ with $j \in \{0,1\}$, or efq_P for some atom P. We will denote the *empty sequence* with ε and with $\xi, \xi', \ldots$ the stacks of length 1. If t is a proof term, $t\,\sigma$ denotes the term $(((t\,\sigma_1)\,\sigma_2) \ldots \sigma_n)$.

We show now that the reductions of the calculus are sound proof transformations.

Theorem 1 (Subject Reduction)**.** *If $t : A$ and $t \mapsto u$, then $u : A$ and all the free variables of u appear among those of t.*

Proof. It is enough to prove the claim for cross reductions. The proof that the intuitionistic reductions and the permutation rules preserve the type is completely standard, see e.g. [9]. Basic cross reductions require straightforward considerations as well. Suppose that

$$(\ldots \parallel \mathscr{C}[\overline{a}\, v] \parallel \ldots) \parallel_a \mathscr{D} \mapsto (\mathscr{C}[\overline{b}\, \langle y\rangle] \parallel_a \mathscr{D}) \parallel_b \mathscr{D}[v^{b/y}/a]$$

Since $\langle y\rangle : B$, then $b^{\neg B}\langle y\rangle : \bot$ and $\mathscr{C}[\overline{b}\, \langle y\rangle]$ can be assigned the correct type. Since the types of $v^{b/y}$ and a are the same, the term $\mathscr{D}[v^{b/y}/a]$ is correctly defined. Finally, since $\overline{a}$ is rightmost in $\mathscr{C}[\overline{a}\, v]$ and y is the sequence of the free variables of t which are bound in $\mathscr{C}[\overline{a}\, v]$, by Def. 3, all free variables of $v^{b/y}$ in $\mathscr{D}[v^{b/y}/a]$ are also free in $\mathscr{C}[\overline{a}\, v]$. Hence, no new free variable is created during the reduction. □

3 The Normalization Theorem

We prove that every proof term of λ_{CL} reduces in a finite number of steps to a normal form. By Subject Reduction this implies the normalization for $\mathbf{NJ} + (\mathrm{EM})$ proofs. The normalization proof is based on the method introduced in [1], adapted here to λ_{CL}, refined and completed by the new Lemma 1 which fills a small gap in the proof for λ_{G}. The idea behind the normalization strategy is to employ a suitable complexity measure for terms $u \parallel_a v$ and, each time a reduction has to be performed, to choose the term of maximal complexity. Since cross reductions can be applied as long as there is a violation of the Subformula Property, the natural approach is to define the complexity measure as a function of some fixed set of formulas, representing the formulas that can be safely used without violating the Subformula Property. We start by defining parallel form and normal form.

Definition 8 (Parallel Form)**.** A **parallel form** is defined inductively as follows: a simply typed λ-term is a parallel form; if u and v are parallel forms, then both $u \parallel_a v$ and $u \parallel v$ are parallel forms.

Definition 9 (Normal Forms and Normalizable Terms)**.**

- A **redex** is a term u such that $u \mapsto v$ for some v and basic reduction of Figure 1. A term t is called a **normal form** or, simply, **normal**, if there is no t' such that $t \mapsto t'$. We define NF to be the set of normal λ_{CL}-terms.
- A sequence, finite or infinite, of proof terms $u_1, u_2, \ldots, u_n, \ldots$ is said to be a reduction of t, if $t = u_1$, and for all i, $u_i \mapsto u_{i+1}$. A proof term u of λ_{CL} is **normalizable** if there is a finite reduction of u whose last term is a normal form.

First established in [1], the following property of simply typed λ-terms is crucial for our normalization proof. It ensures that every bound hypothesis appearing in a normal intuitionistic proof is a strong subformula of one of the premises or a proper subformula of the conclusion. This implies that the types of the new channels generated by cross reductions are smaller than the local premises.

Proposition 2. *Suppose that $t \in \mathsf{NF}$ is a simply typed λ-term, $x_1^{A_1}, \ldots, x_n^{A_n} \vdash t : A$ and $z : B$ is a variable occurring bound in t. Then one of the following holds: (1) B is a proper subformula of a prime factor of A or (2) B is a strong subformula of one among $A_1, \ldots, A_n$.*

As shown in [1], each hypothesis of a normal intuitionistic proof is followed by an elimination rule, unless the hypothesis is $\bot$, subformula of the conclusion or proper subformula of a premise.

Proposition 3. *Let $t \in \mathsf{NF}$ be a simply typed λ-term and $x_1^{A_1}, \ldots, x_n^{A_n}, z^B \vdash t : A$ One of the following holds:*

1. *Every occurrence of z^B in t is of the form $z^B\,\xi$ for some proof term or projection ξ.*
2. *$B = \bot$ or B is a subformula of A or a proper subformula of one among $A_1, \ldots, A_n$.*

Proposition 4 (Parallel Form Property)**.** *If $t \in \mathsf{NF}$ is a λ_{CL}-term, then it is in parallel form.*

Proof. Easy structural induction on t using the permutation reductions. □

Definition 10 (Complexity of Parallel Terms)**.** Let $\mathscr{A}$ be a finite set of formulas. The $\mathscr{A}$**-complexity** of $u \parallel_a v$ is the sequence (c,d,l,o) of natural numbers, where:

1. if the communication kind of a is C, then c is the maximum among 0 and the number of symbols of the prime factors of C that are not subformulas of some formula in $\mathscr{A}$;
2. d is the number of occurrences of $\parallel_e$ and $\parallel$ in u, v for any variable e;
3. l is the sum of the maximal lengths of the intuitionistic reductions of u, v;
4. o is the number of occurrences of a in u, v.

The $\mathscr{A}$**-communication-complexity** of $u \parallel_a v$ is c.

We adapt the normalization algorithm of [1] that represents the constructive content of the proofs of Prop. 5 and Thm. 2. Essentially, the master reduction strategy consists in iterating the basic reduction relation $\succ$ defined in Def. 11 below, whose goal is to permute the smallest redex $u \parallel_a v$ of maximal complexity until u, v are simple parallel terms (see Def. 1) then normalize them and apply cross reductions.

Definition 11 (Side Reduction Strategy)**.** Let $t : A$ be a term with free variables $x_1^{A_1}, \ldots, x_n^{A_n}$ and $\mathscr{A}$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1, \ldots, A_n$. Let $u \parallel_a v$ be the *smallest subterm* of t, if any, among those of *maximal* $\mathscr{A}$-complexity and let (c,d,l,o) be its $\mathscr{A}$-complexity. We write $t \succ t'$ whenever t' has been obtained from t by applying to $u \parallel_a v$:

1. if $d > 0$, a permutation that move $\parallel_a$ inside u or v, such as $u \parallel_a (v_1 \parallel_b v_2) \mapsto (u \parallel_a v_1) \parallel_b (u \parallel_a v_2)$
2. if $d = 0$ and $l > 0$, intuitionistic reductions normalizing all terms $u_1, \ldots, u_m$;
3. if $d = l = 0$ and $c > 0$, a cross reduction possibly followed by applications of the cross reductions $w_1 \parallel_c w_2 \mapsto w_i$ for $i \in \{1,2\}$ to the whole term;
4. if $d = l = c = 0$, a cross reduction $u \parallel_a v \mapsto u$ or $u \parallel_a v \mapsto v$.

Definition 12 (Master Reduction Strategy)**.** We define a normalization algorithm $\mathscr{N}(t)$ which for any term t outputs a term t' such that $t \mapsto^* t'$. Let the free variables of t be $x_1^{A_1}, \ldots, x_n^{A_n}$ and $\mathscr{A}$ be the set of proper subformulas of A and strong subformulas of $A_1, \ldots, A_n$. The algorithm behaves as follows.

1. If t is not in parallel form, then by permutation reductions t is reduced to a t' which is in parallel form and $\mathscr{N}(t')$ is recursively executed.
2. If t is a simply typed λ-term, it is normalized and returned. If $t = u_1 \parallel_a u_2$ is not a redex, then let $\mathscr{N}(u_i) = u_i'$ for $1 \leq i \leq 2$. If $u_1' \parallel_a u_2'$ is normal, it is returned. Otherwise, $\mathscr{N}(u_1' \parallel_a u_2')$ is recursively executed.
3. If t is a redex, we select the *smallest* subterm w of t having maximal $\mathscr{A}$-communication-complexity r. A sequence of terms $w \succ w_1 \succ w_2 \succ \ldots \succ w_n$ is produced such that w_n has $\mathscr{A}$-communication-complexity strictly smaller than r. We replace w by w_n in t, obtain t', and recursively execute $\mathscr{N}(t')$.

We observe that in the step 2 of the algorithm $\mathscr{N}$, by construction $u_1 \parallel_a u_2$ is not a redex. After u_1, u_2 are normalized respectively to u_1', u_2', it can still be the case that $u_1' \parallel_a u_2'$ is not normal, because some free variables of u_1, u_2 may disappear during the normalization, causing a new violation of the Subformula Property that transforms $u_1' \parallel_a u_2'$ into a redex, even though $u_1 \parallel_a u_2$ was not.

The first step of the normalization consists in reducing the term in parallel form.

Proposition 5. *Let $t : A$ be any term. Then $t \mapsto^* t'$, where t' is a parallel form.*

Proof. Easy structural induction on t. □

We now prove that any term in parallel form can be normalized using the algorithm $\mathscr{N}$.

Lemma 1. *Let $t : A$ be a term in parallel form which is not simply typed and $\mathscr{A}$ contain all proper subformulas of A and be closed under subformulas. Assume that $r > 0$ is the maximum $\mathscr{A}$-communication-complexity of the subterms of t. Assume that the free variables $x_1^{A_1}, \dots, x_n^{A_n}$ of t are such that for every i, either each strong subformula of A_i is in $\mathscr{A}$, or each proper prime subformula of A_i is in $\mathscr{A}$ or has at most r symbols. Suppose moreover that no subterm $u_1 \parallel_a u_2$ with $\mathscr{A}$-communication-complexity r contains a subterm of the same $\mathscr{A}$-communication-complexity. Then there exists t' such that $t \succ^* t'$ and the maximal among the $\mathscr{A}$-communication-complexity of the subterms of t' is strictly smaller than r.*

Proof. We prove the lemma by lexicographic induction on the pair (ρ, k) where k is the number of subterms of t with maximal $\mathscr{A}$-complexity ρ among those with $\mathscr{A}$-communication-complexity r.

Let $u_1 \parallel_a u_2$ be the *smallest* subterm of t having $\mathscr{A}$-complexity ρ. Four cases can occur.

(a) $\rho = (r, d, l, o)$, with $d > 0$. We first show that the term $u_1 \parallel_a u_2$ is a redex. Now, the free variables of $u_1 \parallel_a u_2$ are among $x_1^{A_1}, \dots, x_n^{A_n}, a_1^{B_1}, \dots, a_p^{B_p}$ and the communication kind of a is D. Hence, suppose by contradiction that all the prime factors of D are proper subformulas of A or strong subformulas of one among $A_1, \dots, A_n, B_1, \dots, B_p$. Given that $r > 0$ there is a prime factor P of D such that P has r symbols and does not belong to $\mathscr{A}$. The possible cases are two: (i) P is a proper subformula of a prime proper subformula A_i' of A_i such that $A_i' \notin \mathscr{A}$; (ii) P, by Prop. 1, is a proper subformula of a prime factor of B_i. If (i), then the number of symbols of A_i' is less than or equal to r, so P cannot be a proper subfomula of A_i', which is a contradiction. If (ii), then, since by hypothesis $a_i^{B_i}$ is bound in t, there is a prime factor of B_i having a number of symbols greater than r, hence we conclude that there is a subterm $w_1 \parallel_b w_2$ of t having $\mathscr{A}$-complexity greater than ρ, which is absurd.

Now, since $d > 0$, we may assume that for some $1 \le i \le 2$, $u_i = w_1 \parallel_b w_2$. Suppose $i = 2$. The term $u_1 \parallel_a (w_1 \parallel_b w_2)$ is then a redex of t and by replacing it with $(*)$: $(u_1 \parallel_a w_1) \parallel_b (u_1 \parallel_a w_2)$ we obtain from t a term t' such that $t \succ t'$ according to Def. 11. We must verify that we can apply to t' the main induction hypothesis. Indeed, the reduction $t \succ t'$ duplicates all the subterms of v, but all of their $\mathscr{A}$-complexities are smaller than r, because $u_1 \parallel_a u_2$ by choice is the smallest subterm of t having maximal $\mathscr{A}$-complexity ρ. The terms $(u_1 \parallel_a w_i)$ for $1 \le i \le 2$ have smaller $\mathscr{A}$-complexity than ρ, because they have numbers of occurrences of the symbol $\parallel$ strictly smaller than in $u_1 \parallel_a u_2$. Moreover, the terms in t' with $(*)$ as a subterm have, by hypothesis, $\mathscr{A}$-communication-complexity smaller than r and hence $\mathscr{A}$-complexity smaller than ρ. Assuming that the communication kind of b is F, the prime factors of F that are not in $\mathscr{A}$ must have fewer symbols than the prime factors of D that are not in $\mathscr{A}$, again because $u_1 \parallel_a u_2$ by choice is the smallest subterm of t having maximal $\mathscr{A}$-complexity ρ; hence, the $\mathscr{A}$-complexity of $(*)$ is smaller than ρ. Hence the number of subterms of t' with $\mathscr{A}$-complexity ρ is strictly smaller than k. By induction hypothesis, $t' \succ^* t''$, where t'' satisfies the thesis.

(b) $\rho = (r, d, l, o)$, with $d = 0$ and $l > 0$. Since $d = 0$, u_1, u_2 are simple parallel terms – and thus strongly normalizable [9] – so we may assume that for $1 \le i \le 2$, $u_i \mapsto^* u_i' \in \mathsf{NF}$ by a sequence intuitionistic reduction rules. By replacing in t the subterm $u_1 \parallel_a u_2$ with $u'_1 \parallel_a u'_2$, we obtain a term t' such that $t \succ t'$ according to Def. 11. Moreover, the terms in t' with $u'_1 \parallel_a u'_2$ as a subterm have, by hypothesis, $\mathscr{A}$-communication-complexity smaller than r and hence $\mathscr{A}$-complexity is smaller than ρ. By induction hypothesis, $t' \succ^* t''$, where t'' satisfies the thesis.

(c) $\rho = (r,d,l,o)$, with $d = l = 0$. Since $d = 0$, u_1, u_2 are simply typed λ-terms. Since $l = 0$, u_1, u_2 are in normal form and thus satisfy conditions 1. and 2. of Prop. 2. We need to check that $u_1 \parallel_a u_2$is a redex, in particular that the communication complexity of a is greater than 0. Assume that the free variables of $u_1 \parallel_a u_2$ are among $x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{B_1}, \ldots, a_p^{B_p}$ and that the communication kind of a is D. As we argued above, we obtain that not all the prime factors of D are proper subformulas of A or strong subformulas of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$. By Def. 6, $u_1 \parallel_a u_2$ is a redex.

We now prove that every occurrence of a in u_1, u_2 is of the form $a\,\xi$ for some term or projection ξ. First of all, a occurs with arrow type in all u_1, u_2. Moreover, $u_1 : A, u_2 : A$, since $t : A$ and t is a parallel form; hence, the types of the occurrences of a in u_1, u_2 cannot be subformulas of A, otherwise $r = 0$, and cannot be proper subformulas of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$, otherwise the prime factors of D would be strong subformulas of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$ and thus we are done. Thus by Prop. 3 we are done. Two cases can occur.

• a does not occur in u_i for $1 \leq i \leq 2$. By performing a cross reduction, we replace in t the term $u_1 \parallel_a u_2$ with u_i and obtain a term t' such that $t \succ t'$ according to Def. 11. After the replacement, the number of subterms having maximal $\mathscr{A}$-complexity ρ in t' is strictly smaller than the number of such subterms in t. By induction hypothesis, $t' \succ^* t''$, where t'' satisfies the thesis.

• a occurs in all the subterms u_1, u_2. Let $u_1 = (\ldots \parallel \mathscr{C}[\overline{a}\, w] \parallel \ldots)$ where $\overline{a} : \neg D$, $(\ldots \parallel \mathscr{C}[\overline{a}\, w] \parallel \ldots)$ is a normal simple parallel term, $\mathscr{C}[\,]$ is a simple context, and the displayed occurrence of $\overline{a}$ is rightmost in $\mathscr{C}[\overline{a}\, w]$. By applying a cross reduction to $\mathscr{C}[\overline{a}\, w] \parallel_a u_2$ we obtain either the term $u_2[w/a]$ or the term $(**)$ $(\mathscr{C}[\overline{b}\,\langle y\rangle] \parallel_a u_2) \parallel_b u_2[w^{b/y}/a]$ where $\overline{b} : \neg B$, y is the sequence of the free variables of w_z which are bound in $\mathscr{C}[\overline{a}\, w]$ and $\overline{a}$ does not occur in w. In the former case, the term has $\mathscr{A}$-complexity strictly smaller than ρ and we are done. In the latter case, since u_1, u_2 satisfy conditions 1. and 2. of Prop. 2, the types $Y_1, \ldots, Y_k$ of the variables y are proper subformulas of A or strong subformulas of the formulas $A_1, \ldots, A_n, B_1, \ldots, B_p$. Hence, the types among $Y_1, \ldots, Y_k$ which are not in $\mathscr{A}$ are strictly smaller than all the prime factors of the formulas $B_1, \ldots, B_p$. Since the communication kind of b consists of the formulas $Y_1 \wedge \ldots \wedge Y_k$, by Def. 10 the $\mathscr{A}$-complexity of the term $(**)$ above is strictly smaller than the $\mathscr{A}$-complexity ρ of $u_1 \parallel_a u_2$.

Now, since $\mathscr{C}[\overline{a}\, w], u_2$ normal simple parallel terms, $\mathscr{C}[\overline{b}\,\langle y\rangle]$ is normal too and contain fewer occurrences of $\overline{a}$ than $\mathscr{C}[\overline{a}\, w]$ does; hence, the $\mathscr{A}$-complexity of the term $\mathscr{C}[\overline{b}\,\langle y\rangle] \parallel_a u_2$ is strictly smaller than the $\mathscr{A}$-complexity ρ of $u_1 \parallel_a u_2$. Let now t' be the term obtained from t by replacing the term $\mathscr{C}[\overline{a}\, w] \parallel_a u_2$ with $(**)$. By construction $t \succ t'$. Moreover, the terms in t' with $(**)$ as a subterm have, by hypothesis, $\mathscr{A}$-communication-complexity smaller than r and hence $\mathscr{A}$-complexity smaller than ρ. Hence, we can apply the main induction hypothesis to t' and obtain by induction hypothesis, $t' \succ^* t''$, where t'' satisfies the thesis.

(d) $\rho = (r,d,l,o)$, with $d = l = o = 0$. Since $o = 0$, $u_1 \parallel_a u_2$ is a redex. Let us say a does not occur in u_i for $1 \leq i \leq 2$. By performing a cross reduction, we replace $u_1 \parallel_a u_2$ with u_i according to Def. 11. Hence, by induction hypothesis, $t' \succ^* t''$, where t'' satisfies the thesis. □

Theorem 2. *Let $t : A$ be a λ_{CL}-term. Then $t \mapsto^* t' : A$, where t' is a normal parallel form.*

Proof. By Prop. 5, we can assume that $t : A$ is in parallel form. Assume now that the free variables of t are $x_1^{A_1}, \ldots, x_n^{A_n}$ and let $\mathscr{A}$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1, \ldots, A_n$. We prove the theorem by lexicographic induction on the quadruple $(|\mathscr{A}|, r, k, s)$ where $|\mathscr{A}|$ is

the cardinality of $\mathscr{A}$, r is the maximal $\mathscr{A}$-communication-complexity of the subterms of t, k is the number of subterms of t having maximal $\mathscr{A}$-communication-complexity r and s is the size of t. If t is a simply typed λ-term, it has a normal form [9] and we are done; so we assume t is not. There are two main cases.
• *First case*: *t is not a redex.* Let $t = u_1 \parallel_a u_2$ and let C be the communication kind of a. Then, the communication complexity of a is 0 and by Def. 6 every prime factor of C belongs to $\mathscr{A}$. Let the types of the occurrences of a in u_i for $1 \le i \le 2$ be B_i, with $B_i = \neg C$ or $B_i = C$. Let now $\mathscr{A}_i$ be the set of the proper subformulas of A and the strong subformulas of $A_1,\dots,A_n,B_i$. By Prop. 1, every strong subformula of B_i is a proper subformula of a prime factor of C, and this prime factor is in $\mathscr{A}$. Hence, $\mathscr{A}_i \subseteq \mathscr{A}$.

If $\mathscr{A}_i = \mathscr{A}$, the maximal $\mathscr{A}_i$-communication-complexity of the terms of u_i is less than or equal to r and the number of terms with maximal $\mathscr{A}_i$-communication-complexity is less than or equal to k; since the size of u_i is strictly smaller than that of t, by induction hypothesis $u_i \mapsto^* u_i'$, where u_i' is a normal parallel form.

If $\mathscr{A}_i \subset \mathscr{A}$, again by induction hypothesis $u_i \mapsto^* u_i'$, where u_i' is a normal parallel form.

Let now $t' = u_1' \parallel_a u_2'$, so that $t \mapsto^* t'$. If t' is normal, we are done. Otherwise, since u_j' for $1 \le j \le 2$ are normal, the only possible redex remaining in t' is the whole term itself, i.e., $u_1' \parallel_a u_2'$: this happens only if the free variables of t' are fewer than those of t; w.l.o.g., assume they are $x_1^{A_1},\dots,x_i^{A_i}$, with $i < n$. Let $\mathscr{B}$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1,\dots,A_i$. Since t' is a redex, the communication complexity of a is greater than 0; by def. 6, a prime factor of C is not in $\mathscr{B}$, so we have $\mathscr{B} \subset \mathscr{A}$. By I.H., $t' \mapsto^* t''$, where t'' is a parallel normal form.
• *Second case*: *t is a redex.* Let $u_1 \parallel_a u_2$ be the *smallest* subterm of t having $\mathscr{A}$-communication-complexity r. The free variables of $u_1 \parallel_a u_2$ satisfy the hypotheses of Lem. 1 either because have type A_i and $\mathscr{A}$ contains all the strong subformulas of A_i, or because the prime proper subformulas of their type have at most r symbols, by maximality of r. By Lem. 1 $u_1 \parallel_a u_2 \succ^* w$ where the maximal among the $\mathscr{A}$-communication-complexity of the subterms of w is strictly smaller than r. Let t' be the term obtained replacing w for $u_1 \parallel_a u_2$ in t. We apply the I.H. and obtain $t' \mapsto^* t''$ with t'' in parallel normal form. □

We prove now that the Subformula Property holds: a normal proof does not contain concepts that do not already appear in the premises or in the conclusion.

Theorem 3 (Subformula Property)**.** *Suppose $x_1^{A_1},\dots,x_n^{A_n} \vdash t : A$, with $t \in \mathsf{NF}$. Then (i) for each communication variable a occurring bound in t and with communication kind $C_1,\dots,C_m$, the prime factors of $C_1,\dots,C_m$ are proper subformulas of $A_1,\dots,A_n,A$; (ii) the type of any subterm of t which is not a bound communication variable is either a subformula or a conjunction of subformulas of $A_1,\dots,A_n,A$.*

Proof. By induction on t. □

4 On the expressive power of λ_{CL}

We discuss the relative expressive power of λ_{CL} and its computational capabilities.

Comparison with π-calculus and λ_G In contrast with the π-calculus [16, 19] which is a formalism for *modeling* concurrent systems, λ_{CL} is a parallel *functional language* intended (as a base) for programming. The first similarity between the two calculi is in the channel restrictions: the a of $u \parallel_a v$ in λ_{CL} and of $\nu a(P \mid Q)$ in π-calculus have the same rôle. Moreover the result of communicating in λ_{CL} a closed process or data is as in the asynchronous π-calculus [12]. In contrast with the π-calculus whose communication only applies to data or channels, the communication in λ_{CL} is higher-order. Moreover, the latter can handle not only closed and open processes, but also processes that are closed in their original environment, but become open after the communication. The number of recipients of a communication is another difference between π-calculus and λ_{CL}. While in pure π-calculus both sender and recipient of a communication might be selected non-deterministically, in λ_{CL}, since the communication is a broadcasting to all recipients, only the sender, $\mathscr{C}_i[\overline{a}\, t_i]$ for $i \in \{1, \ldots, n\}$ in the following example, can be non-deterministically selected:

$$(\mathscr{C}_1[\overline{a}\, t_1] \parallel \ldots \parallel \mathscr{C}_n[\overline{a}\, t_n]) \parallel_a \mathscr{D} \quad \mapsto \quad \mathscr{D}[t_i/a]$$

Furthermore, in π-calculus only one process can receive each message whereas in λ_{CL} we can have one-to-many communications, or broadcast:

$$\mathscr{C}[\overline{a}\, u] \parallel_a (\mathscr{D}_1 \parallel \ldots \parallel \mathscr{D}_m) \quad \mapsto \quad \mathscr{D}_1[u/a] \parallel \ldots \parallel \mathscr{D}_m[u/a]$$

Finally, while in π-calculus there is no restriction on the use of channels between processes, in λ_{CL} there are strict symmetry conditions; similar conditions are adopted in typed versions of π-calculus (see [21, 23]). Hence λ_{CL} cannot encode a *dialogue* between two processes: if a process u receives a message from a process v, then v cannot send a message to u. To model these exchanges, more complex calculi such as λ_G [1] are needed. If a λ_{CL} channel connects two processes as shown below on the left, a λ_G channel connects them as shown below on the right.

Namely, a λ_{G} channel can transmit messages between the processes in both directions. Even though the communication mechanism of λ_{G} enables us to define unidirectional channels as well, the technical details of λ_{G} and λ_{CL} communications differ considerably since they are tailored, respectively, to the linearity axiom $(A \to B) \vee (B \to A)$ and to EM. In λ_{CL} indeed all occurrences of the receiver's channel are simultaneously replaced by the message, but this is not possible in λ_{G}. As a consequence, λ_{G} cannot implement broadcast communication. Finally, while the closure transmission mechanisms of λ_{G} and λ_{CL} have the same function and capabilities – a version of Example 2 for λ_{G} is presented in [1] – λ_{CL} mechanism is considerably simpler.

We establish first the relation of λ_{CL} with the simply typed λ-calculus and Parigot's λ_μ [17], by proving in particular that λ_{CL}, as λ_{G}, can code the parallel OR. Then we show the use of λ_{CL} closure transmission for code optimization.

Proposition 6. λ_{CL} *is strictly more expressive than simply typed λ-calculus and propositional λ_μ.*

Proof. The simply typed λ-calculus can be trivially embedded into λ_{CL}. The converse does not hold, as λ_{CL} can encode the parallel OR, which is a term $\mathsf{O} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ such that $\mathsf{OFF} \mapsto^* \mathsf{F}$, $\mathsf{O}u\mathsf{T} \mapsto^* \mathsf{T}$, $\mathsf{OT}u \mapsto^* \mathsf{T}$ for every term u. By contrast, as a consequence of Berry's sequentiality theorem (see [2]) there is no parallel OR in simply typed λ-calculus. Assuming to add the boolean type in our calculus, that $\overline{a} : \mathsf{Bool} \wedge S \to \bot$, and that $a : \mathsf{Bool} \wedge S$, "$\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$" is as usual, the λ_{CL} term for such parallel OR is

$$\mathsf{O} := \lambda x^{\mathsf{Bool}}\, \lambda y^{\mathsf{Bool}} (\mathsf{if}\ x\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ \overline{a}\, \langle \mathsf{F}, s\rangle\, \mathsf{efq}_{\mathsf{Bool}} \parallel_a \mathsf{if}\ y\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ a\, \pi_0)$$

for any flag term $s : S$, introduced for a complete control of the reduction. Now, $\mathsf{O}\, u\, \mathsf{T}$ reduces to T by

$$(\mathsf{if}\ u\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ \overline{a}\, \langle \mathsf{F}, s\rangle\, \mathsf{efq}_{\mathsf{Bool}}) \parallel_a (\mathsf{if}\ \mathsf{T}\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ a\, \pi_0) \mapsto^* (\mathsf{if}\ u\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ \overline{a}\, \langle \mathsf{F}, s\rangle\, \mathsf{efq}_{\mathsf{Bool}}) \parallel_a \mathsf{T} \ \mapsto\ \mathsf{T}$$

And symmetrically $\mathsf{O}\,\mathsf{T}\,u \mapsto^* \mathsf{T}$. On the other hand, $\mathsf{O}\,\mathsf{F}\,\mathsf{F}$ reduces to F by

$$(\text{if F then F else } \overline{a}\,\langle \mathsf{F}, s\rangle\,\mathsf{efq}_{\mathsf{Bool}}) \parallel_a (\text{if F then F else } a\,\pi_0) \mapsto^* \overline{a}\,\langle \mathsf{F}, s\rangle\,\mathsf{efq}_{\mathsf{Bool}} \parallel_a a\,\pi_0 \;\mapsto\; \langle \mathsf{F}, s\rangle\,\pi_0 \;\mapsto\; \mathsf{F}$$

The claim follows by Ong's embedding of propositional λ_μ in the simply typed λ-calculus, see Lemma 6.3.7 of [20]. Indeed the translation $\underline{u}$ of a $\lambda\mu$-term u is such that $\underline{st} = \underline{s}\,\underline{t}$ and $\underline{x} = x$ for any variable x, if there were a typed $\lambda\mu$-term O for parallel OR, then

$$\underline{\mathsf{O}}\,x\,\underline{\mathsf{T}} = \underline{\mathsf{O}\,x\,\mathsf{T}} \quad\mapsto^*\quad \underline{\mathsf{T}} = \mathsf{T} \qquad \underline{\mathsf{O}}\,\underline{\mathsf{T}}\,x = \underline{\mathsf{O}\,\mathsf{T}\,x} \quad\mapsto^*\quad \underline{\mathsf{T}} = \mathsf{T} \qquad \underline{\mathsf{O}}\,\underline{\mathsf{F}}\,\underline{\mathsf{F}} = \underline{\mathsf{O}\,\mathsf{F}\,\mathsf{F}} \quad\mapsto^*\quad \underline{\mathsf{F}} = \mathsf{F}$$

and $\underline{\mathsf{O}}$ would be a parallel OR in simply typed λ-calculus, which is impossible. □

Example 1 (**Classical Disjunction**). Since in classical logic disjunction is definable, the corresponding computational constructs of case distinction and injection can be defined in λ_{CL}. By contrast, these constructs are usually added as new primitives in simply typed λ-calculus, as simulating them requires complicated CPS-translations.

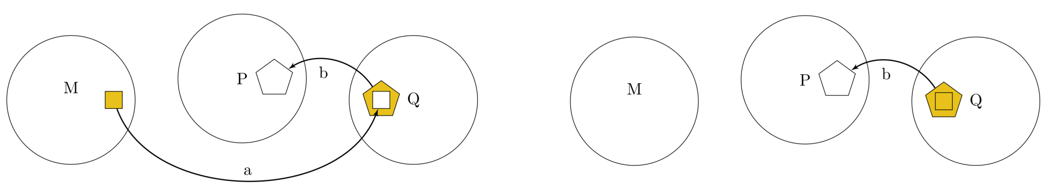
The λ_{CL} terms $\iota_0(u)$, $\iota_1(u)$ and $t[x_0.v_0, x_1.v_1]$ such that for $i \in \{0,1\}$ we have $\iota_i(u)[x_0.v_0, x_1.v_1] \mapsto v_i[u/x_i]$ are defined as follows: Let $A \vee B := (A \to \bot) \to (B \to \bot) \to \bot$

$$\iota_0(u) \;:=\; \lambda x^{A\to\bot}\lambda y^{B\to\bot}\,x\,u : A \vee B \qquad\qquad \iota_1(u) \;:=\; \lambda x^{A\to\bot}\lambda y^{B\to\bot}\,y\,u : A \vee B$$
$$t\,[x_0.v_0, x_1.v_1] \;:=\; (\,\mathsf{efq}_F(t\,\overline{a}\,\overline{b}\,) \parallel_a v_0[a/x_0]) \parallel_b v_1[b/x_1] : F$$

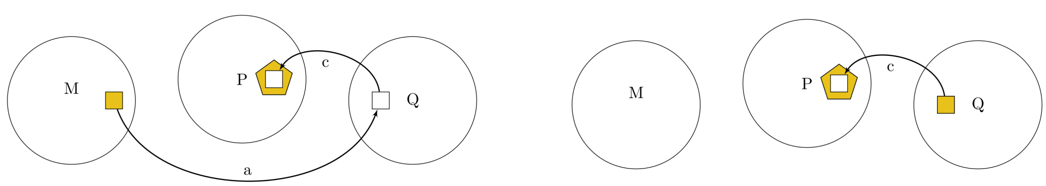
where $a : A$, $\overline{a} : A \to \bot$, $b : B$, $\overline{b} : B \to \bot$, $v_0 : F$, $v_1 : F$, $t : A \vee B$ and efq_F is a closed term of type $\bot \to F$. We can then verify, for example, that

$$\iota_0(u)\,[x_0.v_0, x_1.v_1] \;:=\; (\,\mathsf{efq}_F((\lambda x^{A\to\bot}\lambda y^{B\to\bot}\,x\,u)\,\overline{a}\,\overline{b}) \parallel_a v_0[a/x_0]) \parallel_b v_1[b/x_1]$$
$$\mapsto^* \;(\,\mathsf{efq}_F(\overline{a}\,u) \parallel_a v_0[a/x_0]) \parallel_b v_1[b/x_1] \;\mapsto\; v_0[u/x_0] \parallel_b v_1[b/x_1] \;\mapsto\; v_0[u/x_0]$$

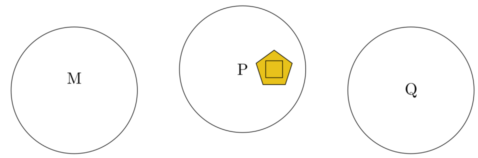
Example 2 (**Cross reductions for program efficiency**). We show how to use cross reductions to communicate processes that are still waiting for some arguments. Consider the process $M \parallel_a (Q \parallel_b P)$. The process Q contains a channel b to send a message (yellow pentagon) to P (below left), but the message is missing a part (yellow square) which is computed by M and sent to Q by a. In a system without a closure handling mechanism, the whole interaction needs to wait until M can communicate to Q (below right).



The cross reduction handles precisely this kind of missing arguments. It enables Q to send immediately the message through the channel a and establishes a new communication channel c on the fly (below left) which redirects the missing term, when ready, to the new location of the message inside P (below right).



We can now partially evaluate P, which in the best case will not even need the yellow square.

Both reductions terminate then with 

the former, sending the whole message (yellow pentagon and square) by b; the latter, redirecting the missing part of the message (yellow square) by the new channel c. For a concrete example assume that

$$M \mapsto^* (\overline{a}\,(\lambda x^{T\to\bot}\, xt))\,\mathsf{efq}_S \qquad Q = (a(\lambda y^T\, \overline{b}\,\langle s,y\rangle))\,\mathsf{efq}_S \qquad P = b\pi_0$$

where $s : S$ and $t : T$ are closed terms, the complexity of S is much higher than that of T, $b : S \wedge T, \overline{b} : \neg(S\wedge T), a : (T \to \bot) \to \bot$ and $\overline{a} : \neg((T\to\bot)\to\bot)$. Without a special mechanism for sending open terms, Q must wait for M to normalize. Afterwards M sends $\lambda x^{T\to\bot} xt$ by a to Q:

$$M \parallel_a (Q \parallel_b P) \overset{\text{since } a \text{ is not in } P}{\mapsto^*} (M \parallel_a Q) \parallel_b P \mapsto^* ((\overline{a}\,(\lambda x^{T\to\bot}\, xt))\,\mathsf{efq}_S \parallel_a Q) \parallel_b P \quad \mapsto$$
$$((\lambda x^{T\to\bot}\, xt)(\lambda y^T\, \overline{b}\,\langle s,y\rangle))\,\mathsf{efq}_S \parallel_b P \mapsto ((\lambda y^T\, \overline{b}\,\langle s,y\rangle)t)\,\mathsf{efq}_S \parallel_b P \mapsto (\overline{b}\,\langle s,t\rangle)\,\mathsf{efq}_S \parallel_b b\pi_0 \mapsto \langle s,t\rangle\pi_0 \mapsto s$$

Clearly P does not need t at all. Even though it waited for the pair $\langle s,t\rangle$, P only uses the term s.

Our normalization instead enables Q to directly send $\langle s,y\rangle$ to P by executing a full cross reduction:

$$M \parallel_a (Q \parallel_b P) = M \parallel_a ((a(\lambda y^T\, \overline{b}\,\langle s,y\rangle))\,\mathsf{efq}_S \parallel_b b\pi_0) \quad \mapsto^* \quad M \parallel_a (((a(\lambda y^T\, \overline{c}\, y))\,\mathsf{efq}_S \parallel_b P) \parallel_c \langle s,c\rangle\pi_0)$$

where the channel c handles the redirection of the data y^T in case it is available later. In our case P already contains all it needs to terminate its computation, indeed

$$\mapsto \quad M \parallel_a (((a(\lambda y^T\, \overline{c}\, y))\,\mathsf{efq}_S \parallel_b P) \parallel_c s) \quad \mapsto^* \quad s$$

since s does not contain communications anymore. Notice that the time-consuming normalization of the term M does not even need to be finished at this point.

Conclusions

We introduced λ_{CL}, a parallel extension of simply typed λ-calculus. The calculus λ_{CL} provides a first computational interpretation of classical proofs as *parallel* programs. Our calculus is defined via Curry–Howard correspondence using a natural deduction system based on the EM axiom $A \vee \neg A$. The definition of λ_{CL} exploits ideas and techniques developed in [1] for the calculus λ_{G} based on the linearity axiom, but the specific features of EM made it possible to define a significantly simpler calculus with more manageable reductions – including those for the transmission of closures. In spite of its simplicity, the resulting calculus is more expressive than simply typed λ-calculus and Parigot's λ_μ [17]. Furthermore terms typed by (EM) admit communication reductions including broadcast communications and races.

Finally, we remark that the permutation reductions of parallel operators undermine a strong normalization result for the calculus. Indeed, such reductions enable loops similar to those occurring in cut-elimination procedures for sequent calculi. Restrictions on the permutations might be enough to prove strong normalization, but we leave this as an open problem.

References

[1] F. Aschieri, A. Ciabattoni & F.A. Genco (2017): *Gödel logic: From natural deduction to parallel computation*. In: *LICS 2017*, pp. 1–12, doi:10.1109/LICS.2017.8005076.

[2] H.P. Barendregt (1984): *The Lambda Calculus, its Syntax and Semantics*. Amsterdam: North-Holland, doi:10.1016/c2009-0-14341-6.

[3] G. Boudol (1989): *Towards a lambda-calculus for concurrent and communicating systems*. In: *TAPSOFT 1998*, pp. 149–161, doi:10.1007/3-540-50939-9_130.

[4] P.-L. Curien & H. Herbelin (2000): *The duality of computation*. In: *ICFP 2000*, pp. 233–243, doi:10.1145/351240.351262.

[5] V. Danos & J.-L. Krivine (2000): *Disjunctive Tautologies as Synchronisation Schemes*. *CSL 2000*, pp. 292–301, doi:10.1007/3-540-44622-2_19.

[6] C. Ene & T. Muntean (1999): *Expressiveness of point-to-point versus broadcast communications*. In: *FCT 1999*, pp. 258–268, doi:10.1007/3-540-48321-7_21.

[7] J. Epstein, A.P. Black & S.L. Peyton Jones (2011): *Towards Haskell in the cloud*. In: *ACM Haskell Symposium 2011*, pp. 118–129, doi:10.1145/2034675.2034690.

[8] A. Fuggetta, G.P. Picco & G. Vigna (1998): *Understanding Code Mobility*. *IEEE Trans. Software Eng.* 24(5), pp. 342–361, doi:10.1109/32.685258.

[9] J.-Y. Girard, Y. Lafont & P. Taylor (1989): *Proofs and Types*. Cambridge University Press. Available at `http://www.paultaylor.eu/stable/prot.pdf`.

[10] T.G. Griffin (1990): *A Formulae-as-Type Notion of Control*. In: *POPL 1990*, pp. 47–58, doi:10.1145/96709.96714.

[11] P. de Groote (1995): *A Simple Calculus of Exception Handling*. In: *TLCA 1995*, pp. 201–215, doi:10.1007/BFb0014054.

[12] K. Honda & M. Tokoro (1991): *An Object Calculus for Asynchronous Communication*. In: *ECOOP 1991*, pp. 133–147, doi:10.1007/BFb0057019.

[13] W.A. Howard (1980): *The formulae-as-types notion of construction*. In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, pp. 479–491.

[14] J.-L. Krivine (1990): *Lambda-calcul types et modèles*. In: *Studies in Logic and Foundations of Mathematics*, Masson, pp. 1–176.

[15] J.-L. Krivine (2009): *Realizability in classical logic*. *Panoramas et synthèses*, pp. 197–229. Available at `https://hal.archives-ouvertes.fr/hal-00154500`.

[16] R. Milner (1992): *Functions as Processes*. *Mathematical Structures in Computer Science* 2(2), pp. 119–141, doi:10.1017/S0960129500001407.

[17] M. Parigot (1997): *Proofs of Strong Normalization for Second-Order Classical Natural Deduction*. *J. Symbolic Logic* 62(4), pp. 1461–1479, doi:10.2307/2275652.

[18] D. Prawitz (1971): *Ideas and Results in Proof Theory*. In: *Proceedings of the Second Scandinavian Logic Symposium*, pp. 235–307, doi:10.2307/2271904.

[19] D. Sangiorgi & D. Walker (2003): *The pi-calculus: a Theory of Mobile Processes*. Cambridge University Press.

[20] M.H.B. Sørensen & P. Urzyczyn (1998): *Lectures on the Curry-Howard Isomorphism*. Elsevier, doi:10.1016/s0049-237x(06)80005-4.

[21] B. Toninho, L. Caires & F. Pfenning (2013): *Higher-Order processes, functions, and sessions: a monadic integration*. In: *ESOP 2013*, pp. 350–369, doi:10.1007/978-3-642-37036-6_20.

[22] P. Wadler (2003): *Call-by-value is dual to call-by-name*. *SIGPLAN Notices* 38(9), pp. 189–201, doi:10.1145/944746.944723.

[23] P. Wadler (2012): *Propositions as Sessions*. *J. of Functional Programming* 24, pp. 384–418, doi:10.1145/2398856.2364568.

[24] P. Wadler (2015): *Propositions as Types*. *Communications of the ACM* 58(12), pp. 75–84, doi:10.1145/2699407.