Gödel Logic: from Natural Deduction to Parallel Computation

Federico Aschieri
Institute of Discrete Mathematics and Geometry
TU Wien, Austria

Agata Ciabattoni
Theory and Logic Group
TU Wien, Austria

Francesco A. Genco
Theory and Logic Group
TU Wien, Austria

***Abstract*—Propositional Gödel logic G extends intuitionistic logic with the non-constructive principle of linearity $(A \to B) \vee (B \to A)$. We introduce a Curry–Howard correspondence for G and show that a simple natural deduction calculus can be used as a typing system. The resulting functional language extends the simply typed λ-calculus via a synchronous communication mechanism between parallel processes, which increases its expressive power. The normalization proof employs original termination arguments and proof transformations implementing forms of code mobility. Our results provide a computational interpretation of G, thus proving A. Avron's 1991 thesis.**

I. Introduction

Logical proofs are static. Computations are dynamic. It is a striking discovery that the two coincide: formulas correspond to types in a programming language, logical proofs to programs of the corresponding types and removing detours from proofs to evaluation of programs. This correspondence, known as Curry–Howard isomorphism, was first discovered for constructive proofs, and in particular for intuitionistic natural deduction and typed λ-calculus [20] and later extended to classical proofs, despite their use of non-constructive principles, such as the excluded middle [18], [2] or reductio ad absurdum [17], [30].

Nowadays various different logics (linear [8], modal [28] ...) have been related to many different notions of computation; the list is long, and we refer the reader to [34].

Gödel logic, Avron's conjecture and previous attempts

Twenty-five years have gone by since Avron conjectured in [3] that Gödel logic $\mathbf{G}$ [16] – one of the most useful and interesting logics intermediate between intuitionistic and classical logic – might provide a basis for parallel λ-calculi. Despite the interest of the conjecture and despite various attempts, no Curry–Howard correspondence has so far been provided for $\mathbf{G}$. The main obstacle has been the lack of an adequate natural deduction calculus. Well designed natural deduction inferences can indeed be naturally interpreted as program instructions, in particular as typed λ-terms. Normalization [32], which corresponds to the execution of the resulting programs, can then be used to obtain proofs only containing formulas that are subformulas of some of the hypotheses or of the conclusion. However the problem of finding a natural deduction for $\mathbf{G}$ with this property, called analyticity, looked hopeless for decades.

Supported by FWF: grants M 1930–N35, Y544-N2, and W1255-N23.

All approaches explored so far to provide a precise formalization of $\mathbf{G}$ as a logic for parallelism, either sacrificed analyticity [1] or tried to devise forms of natural deduction whose structures mirror *hypersequents* – which are sequents operating in parallel [4]. Hypersequents were indeed successfully used in [3] to define an analytic calculus for $\mathbf{G}$ and were intuitively connected to parallel computations: the key rule introduced by Avron to capture the linearity axiom – called *communication* – enables sequents to exchange their information and hence to "communicate". The first analytic natural deduction calculus proposed for $\mathbf{G}$ [5] uses indeed parallel intuitionistic derivations joined together by the hypersequent separator. Normalization is obtained there only by translation into Avron's calculus: no reduction rules for deductions and no corresponding λ-calculus were provided. The former task was carried out in [6], that contains a propositional hyper natural deduction with a normalization procedure. The definition of a corresponding λ-calculus and Curry–Howard correspondence are left as an open problem, which might have a complex solution due to the elaborated structure of hyper deductions. Another attempt along the "hyper line" has been made in [19]. However, not only the proposed proof system is not shown to be analytic, but the associated λ-calculus is not a Curry–Howard isomorphism: the computation rules of the λ-calculus are not related to proof transformations, i.e. *Subject Reduction* does not hold.

λ_{G}*: Our Curry–Howard Interpretation of Gödel Logic*

We introduce a natural deduction and a Curry–Howard correspondence for propositional $\mathbf{G}$. We add to the λ-calculus an operator that, from the programming viewpoint, represents parallel computations and communications between them; from the logical viewpoint, the linearity axiom; and from the proof theory viewpoint, the hypersequent separator among sequents. We call the resulting calculus λ_{G}: parallel λ-calculus for $\mathbf{G}$. λ_{G} relates to the natural deduction $\mathbf{NG}$ for $\mathbf{G}$ as typed λ-calculus relates to the natural deduction $\mathbf{NJ}$ for intuitionistic logic $\mathbf{IL}$:

$$\begin{tikzcd}
\mathbf{IL} \arrow[r, leftrightarrow, "\textit{Soundness and}"'] & \mathbf{NJ} \arrow[r, Leftrightarrow, "\textit{Curry–Howard}"'] & \lambda \\
\mathbf{G} \arrow[r, leftrightarrow, "\textit{Completeness}"] & \mathbf{NG} \arrow[r, Leftrightarrow, "\textit{correspondence}"] & \lambda_{\mathrm{G}}
\end{tikzcd}$$

We prove: the perfect match between computation steps and proof reductions in the Subject Reduction Theorem; the Normalization Theorem, by providing a terminating reduction strategy for λ_{G}; the Subformula Property, as corollary. The expressive power of λ_{G} is illustrated through examples of programs and connections with the π-calculus [26], [33].

The natural deduction calculus $\mathbf{NG}$ that we use as type system for λ_{G} is particularly simple: it extends $\mathbf{NJ}$ with the (com) rule (its typed version is displayed below), which was first considered in [24] to define a natural deduction calculus for $\mathbf{G}$, but with no normalization procedure. The calculus $\mathbf{NG}$ follows the basic principle of natural deduction that *new axioms require new computational reductions*; this contrasts with the basic principle of sequent calculus employed in the "hyper approach", that *new axioms require new deduction structures*. Hence we keep the calculus simple and deal with the complexity of the hypersequent structure at the operational side. Consequently, the programs corresponding to $\mathbf{NG}$ proofs maintain the syntactical simplicity of λ-calculus. The normalization procedure for $\mathbf{NG}$ extends Prawitz's method with ideas inspired by hypersequent cut-elimination, by normalization in classical logic [2] and by the embedding in [10] between hypersequents and systems of rules [29]; the latter shows that (com) reformulates Avron's communication rule.

The inference rules of $\mathbf{NG}$ are decorated with λ_{G}-terms, so that we can directly read proofs as typed programs. The decoration of the $\mathbf{NJ}$ inferences is standard and the typed version of (com) is

$$\dfrac{\begin{matrix}[a^{A\to B} : A \to B] \\ \vdots \\ u : C\end{matrix} \qquad \begin{matrix}[a^{B\to A} : B \to A] \\ \vdots \\ v : C\end{matrix}}{u \parallel_a v : C}\;\mathsf{com}$$

Inspired by [1], we use the variable a to represent a *private* communication channel between the processes u and v. The computational reductions associated to $\parallel_a$ – *cross reductions* – enjoy a natural interpretation in terms of higher-order process passing, a feature which is not directly rendered through communication by reference [31] and is also present in higher-order π-calculus [33]. Nonetheless cross reductions handle more subtle migration issues. In particular, a cross reduction can be activated whenever a communication channel a is ready to transfer information between two parallel processes:

$$\mathcal{C}[a\, u] \parallel_a \mathcal{D}[a\, v]$$

Here $\mathcal{C}$ is a process containing a fragment of code u, and $\mathcal{D}$ is a process containing a fragment of code v. Moreover, $\mathcal{C}$ has to send u through the channel a to $\mathcal{D}$, which in turn needs to send v through a to $\mathcal{C}$. In general we cannot simply send the programs u and v: some resources in the computational environment that are used by u and v may become inaccessible from the new locations [14]. Cross reductions solve the problem by exchanging the location of u and v and creating a new communication channel for their resources. Technically, the channel takes care of *closures* – the contexts containing the definitions of the variables used in a function's body [23]. Several programming languages such as JavaScript, Ruby or Swift provide mechanisms to support and handle closures. In our case, they are the basis of a *process migration mechanism* handling the bindings between code fragments and their computational environments. Cross reductions also improve the efficiency of programs by facilitating partial evaluation of open processes (see Example VII.4).

II. Preliminaries on Gödel logic

Also known as Gödel–Dummett logic [12], Gödel logic $\mathbf{G}$ naturally turns up in a number of different contexts; among them, due to the natural interpretation of its connectives as functions over the real interval $[0, 1]$, $\mathbf{G}$ is one of the best known 'fuzzy logics', e.g. [25].

Although propositional $\mathbf{G}$ is obtained by adding the linearity axiom (lin) $(A \to B) \vee (B \to A)$ to any proof calculus for intuitionistic logic, analytic calculi for $\mathbf{G}$ have only been defined in formalisms *extending* the sequent calculus. Among them, arguably, the hypersequent calculus in [3] is the most successful one, see, e.g., [25]. In general a hypersequent calculus is defined by incorporating a sequent calculus (Gentzen's *LJ*, in case of $\mathbf{G}$) as a sub-calculus and allowing sequents to live in the context of finite multisets of sequents.

Definition II.1. A **hypersequent** is a multiset of sequents, written as $\Gamma_1 \Rightarrow \Pi_1 \mid \ldots \mid \Gamma_n \Rightarrow \Pi_n$ where, for all $i = 1, \ldots n$, $\Gamma_i \Rightarrow \Pi_i$ is an ordinary sequent.

The symbol "|" is a meta-level disjunction; this is reflected by the presence in the calculus of the external structural rules of weakening and contraction, operating on whole sequents, rather than on formulas. The hypersequent design opens the possibility of defining new rules that allow the "exchange of information" between different sequents. It is this type of rules which increases the expressive power of hypersequent calculi compared to sequent calculi. The additional rule employed in Avron's calculus for $\mathbf{G}$ [3] is the so called *communication rule*, below presented in a slightly reformulated version (as usual G stands for a possibly empty hypersequent):

$$\frac{G \mid \Gamma_1, B \Rightarrow C \quad G \mid \Gamma_2, A \Rightarrow D}{G \mid \Gamma_1, A \Rightarrow C \mid \Gamma_2, B \Rightarrow D}$$

III. Natural Deduction

The very first step in the design of a Curry–Howard correspondence is to lay a solid logical foundation. No architectural mistake is allowed at this stage: the natural deduction must be structurally simple and the reduction rules as elementary as possible. We present such a natural deduction system $\mathbf{NG}$ for Gödel logic. $\mathbf{NG}$ extends Gentzen's propositional natural deduction $\mathbf{NJ}$ (see [32]) with a rule accounting for axiom (lin). We describe the reduction rules for transforming every $\mathbf{NG}$ deduction into an analytic one and present the ideas behind the Normalization Theorem, which is proved in the λ-calculus framework in Section VI.

$\mathbf{NG}$ is the natural deduction version of the sequent calculus with systems of rules in [29]; the latter embeds (into) Avron's hypersequent calculus for $\mathbf{G}$. Indeed [10] introduces a mapping from (and into) derivations in Avron's calculus into (and from) derivations in the *LJ* sequent calculus for intuitionistic logic with the addition of the system of rules

$$\begin{array}{ccc} \dfrac{B, \Gamma_1 \Rightarrow C}{A, \Gamma_1 \Rightarrow C}\ (com_1) & & \dfrac{A, \Gamma_2 \Rightarrow D}{B, \Gamma_2 \Rightarrow D}\ (com_2) \\ \vdots & & \vdots \\ \multicolumn{3}{c}{\dfrac{\Gamma \Rightarrow \Pi \qquad\qquad \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi}\ (com_{end})} \end{array}$$

where $(com_1), (com_2)$ can only be applied (possibly many times) above respectively the left and right premise of (com_{end}). The above system, that reformulates Avron's *communication* rule, immediately translates into the natural deduction rule below, whose addition to **NJ** leads to a natural deduction calculus for **G**

$$\dfrac{\begin{array}{c} \vdots \\ \dfrac{B}{A}\ \mathsf{com}_l \\ \vdots \\ C \end{array} \qquad \begin{array}{c} \vdots \\ \dfrac{A}{B}\ \mathsf{com}_r \\ \vdots \\ C \end{array}}{C}\ \mathsf{com}$$

Not all the branches of a derivation containing the above rule are **NJ** derivations. To avoid that, and to keep the proof of the Subformula Property (Theorem V.4) as simple as possible, we use the equivalent rule below, first considered in [24].

Definition III.1 (NG)**.** The natural deduction calculus **NG** extends **NJ** with the (com) rule:

$$\dfrac{\begin{array}{c} [A \to B] \\ \vdots \\ C \end{array} \qquad \begin{array}{c} [B \to A] \\ \vdots \\ C \end{array}}{C}\ \mathsf{com}$$

Let $\vdash_{\mathbf{NG}}$ and $\vdash_{\mathbf{G}}$ indicate the derivability relations in **NG** and in $\mathbf{NJ} + (lin)$, respectively.

Theorem III.1 (Soundness and Completeness)**.** *For any set* Π *of formulas and formula* A*,* $\Pi \vdash_{\mathbf{NG}} A$ *if and only if* $\Pi \vdash_{\mathbf{G}} A$*.*

Proof. $(\Rightarrow)$ Applications of (com) can be simulated by $\vee$ eliminations having as major premiss an instance of (lin). $(\Leftarrow)$ Easily follows by the following derivation:

$$\dfrac{\dfrac{[A \to B]^1}{(A \to B) \vee (B \to A)} \qquad \dfrac{[B \to A]^1}{(A \to B) \vee (B \to A)}}{(A \to B) \vee (B \to A)}\ \mathsf{com}^1$$

□

Notation. To shorten derivations henceforth we will use

$$\dfrac{A}{B}\ \mathsf{com}_l \qquad \dfrac{B}{A}\ \mathsf{com}_r \qquad \text{as abbreviations for}$$

$$\dfrac{[A \to B] \qquad A}{B} \qquad \dfrac{[B \to A] \qquad B}{A}$$

respectively, and call them **communication inferences**.

As usual, we will use $\neg A$ and $\top$ as shorthand for $A \to \bot$ and $\bot \to \bot$. Moreover, we exploit the equivalence of $A \vee B$ and $((A \to B) \to B) \ \wedge \ ((B \to A) \to A)$ in **G** (see [12]) and treat $\vee$ as a defined connective.

A. Reduction Rules and Normalization

A normal deduction in **NG** should have two essential features: every intuitionistic Prawitz-style reduction should have been carried out and the Subformula Property should hold. Due to the (com) rule, the former is not always enough to guarantee the latter. Here we present the main ideas behind the normalization procedure for **NG** and the needed reduction rules. The computational interpretation of the rules will be carried out through the λ_{G} calculus in Section IV.

The main steps of the normalization procedure are as follows:

- We permute down all applications of (com).

The resulting deduction – we call it in *parallel form* – consists of purely intuitionistic subderivations joined together by consecutive (com) inferences occurring immediately above the root. This transformation is a key tool in the embedding between hypersequents and systems of rules [10]. The needed reductions are instances of Prawitz-style permutations for $\vee$ elimination. Their list can be obtained by translating into natural deduction the permutations in Fig. 1.

Once obtained a parallel form, we interleave the following two steps.

- We apply the standard intuitionistic reductions ([32]) to the parallel branches of the derivation.

This way we normalize each single intuitionistic derivation, and this can be done in parallel. The resulting derivation, however, need not satisfy yet the Subformula Property. Intuitively, the problem is that communications may discharge hypotheses that have nothing to do with their conclusion.

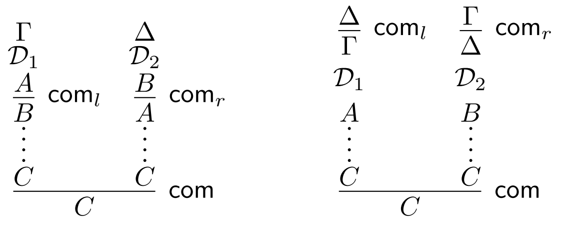
- We apply specific reductions to replace the (com) applications that violate the Subformula Property.

These reductions – called *cross reductions* – account for the hypersequent cut-elimination. They allow to get rid of the new detours that appear in configurations like the one below on the left. To remove these detours, a first idea would be to simultaneously move the deduction $\mathcal{D}_1$ to the right and $\mathcal{D}_2$ to the left thus obtaining the derivation below right:

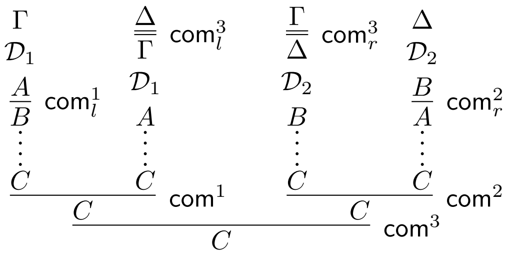
$$\dfrac{\begin{array}{c} \mathcal{D}_1 \\ \dfrac{A}{B}\ \mathsf{com}_l \\ \vdots \\ C \end{array} \qquad \begin{array}{c} \mathcal{D}_2 \\ \dfrac{B}{A}\ \mathsf{com}_r \\ \vdots \\ C \end{array}}{C}\ \mathsf{com} \qquad\qquad \dfrac{\begin{array}{c} \mathcal{D}_2 \\ B \\ \vdots \\ C \end{array} \quad \begin{array}{c} \mathcal{D}_1 \\ A \\ \vdots \\ C \end{array}}{C}$$

In fact, in the context of Krivine's realizability, Danos and Krivine [9] studied the linearity axiom as a theorem of classical logic and discovered that its realizers implement a *restricted* version of this transformation. Their transformation does not lead however to the subformula property for NG. The unrestricted transformation above, on the other hand, cannot work; indeed $\mathcal{D}_1$ might contain the hypothesis $A \to B$ and hence it cannot be moved on the right. Even worse, $\mathcal{D}_1$ may depend on hypotheses that are locally opened, but discharged below B but above C. Again, it is not possible to move $\mathcal{D}_1$ on the right as naively thought, otherwise new global hypotheses would be created.

We overcome these barriers by our cross reductions. Let us highlight Γ and Δ, the hypotheses of $\mathcal{D}_1$ and $\mathcal{D}_2$ that are respectively discharged below B and A but above the application of (com). Assume moreover, that $A \to B$ does not occur in $\mathcal{D}_1$ and $B \to A$ does not occur in $\mathcal{D}_2$ as hypotheses discharged by (com). A cross reduction transforms the deduction below left into the deduction below right (if (com) in the original proof discharges in each branch exactly one occurrence of the hypotheses, and Γ and Δ are formulas)



and into the following deduction, in the general case



where the double bar notation stands for an application of (com) between *sets of hypotheses* Γ and Δ, which means to prove from Γ the conjunction of the formulas of Γ, then to prove the conjunction of the formulas of Δ by means of a communication inference and finally obtain each formula of Δ by a series of $\wedge$ eliminations, and vice versa.

Mindless applications of the cross reductions might lead to dangerous loops, see e.g. Example IV.2. To avoid them we will allow cross reductions to be performed only when the proof is not analytic. Thanks to this and to other restrictions, we will prove termination and thus the Normalization Theorem.

IV. The λ_{G}-Calculus

We introduce λ_{G}, our parallel λ-calculus for $\mathbf{G}$. λ_{G} extends the standard Curry–Howard correspondence [34] for intuitionistic natural deduction with a parallel operator that interprets the inference for the linearity axiom. We describe λ_{G}-terms and their computational behavior, proving as main result of the section the Subject Reduction Theorem, stating that the reduction rules preserve the type.

Axioms $\quad x^A : A$

Conjunction $\quad \dfrac{u : A \quad t : B}{\langle u, t\rangle : A \wedge B} \qquad \dfrac{u : A \wedge B}{u\,\pi_0 : A} \qquad \dfrac{u : A \wedge B}{u\,\pi_1 : B}$

Implication $\quad \dfrac{\begin{array}{c}[x^A : A]\\ \vdots\\ u : B\end{array}}{\lambda x^A u : A \to B} \qquad \dfrac{t : A \to B \quad u : A}{tu : B}$

Linearity Axiom $\quad \dfrac{\begin{array}{cc}[a^{A\to B} : A \to B] & [a^{B\to A} : B \to A]\\ \vdots & \vdots\\ u : C & v : C\end{array}}{u \parallel_a v : C}$

Ex Falso Quodlibet $\quad \dfrac{\Gamma \vdash u : \bot}{\Gamma \vdash \mathsf{efq}_P(u) : P}$ with P atomic, $P \neq \bot$.

The table above defines a type assignment for λ_{G}-terms, called **proof terms** and denoted by $t, u, v \ldots$, which is isomorphic to $\mathbf{NG}$. The typing rules for axioms, implication, conjunction and ex-falso-quodlibet are standard and give rise to the simply typed λ-calculus, while parallelism is introduced by the rule for the linearity axiom.

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 as x^A, y^A, $z^A, \ldots,$ a^A, b^A, c^A and whenever the type is not important simply as $x, y, z, \ldots, a, b$. For clarity, the variables introduced by the (com) rule will be often denoted with letters $a, b, c, \ldots$, 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 **communication variable** and represents a *private* communication channel between the parallel processes u and v.

The free and bound variables of a proof term are defined as usual and for the new term $u \parallel_a v$, all the free occurrences of a in u and v are bound in $u \parallel_a v$. In the following we assume the standard renaming rules and alpha equivalences that are used to avoid capture of variables in the reduction rules.

Notation. The connective $\to$ associates to the right and by $\langle t_1, t_2, \ldots, t_n\rangle$ we denote the term $\langle t_1, \langle t_2, \ldots \langle t_{n-1}, t_n\rangle \ldots\rangle\rangle$ and by π_i, for $i = 0, \ldots, n$, the sequence of projections $\pi_1 \ldots \pi_1 \pi_0$ selecting the $(i+1)$th element of the sequence. Therefore, for every formula sequence $A_1, \ldots, A_n$ the expression $A_1 \wedge \ldots \wedge A_n$ denotes $(A_1 \wedge (A_2 \wedge \ldots (A_{n-1} \wedge A_n) \ldots))$ or $\top$ if $n = 0$.

Often, when $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and the list $x_1, \ldots, x_n$ includes all the free variables of a proof term $t : A$, we shall write $\Gamma \vdash t : A$. From the logical point of view, t represents a natural deduction of A from the hypotheses $A_1, \ldots, A_n$. We shall write $\mathbf{G} \vdash t : A$ whenever $\vdash t : A$, and the notation means provability of A in propositional Gödel logic. If the symbol $\parallel$ does not occur in it, then t is a **simply typed λ-term** representing an intuitionistic deduction.

We define as usual the notion of context $\mathcal{C}[\ ]$ as the part of a proof term that surrounds a hole, represented by some fixed variable. In the expression $\mathcal{C}[u]$ we denote a particular occurrence of a subterm u in the whole term $\mathcal{C}[u]$. We shall just need those particularly simple contexts which happen to be simply typed λ-terms.

Definition IV.1 (Simple Contexts)**.** A **simple context** $\mathcal{C}[\ ]$ is a simply typed λ-term with some fixed variable $[]$ occurring exactly once. For any proof term u of the same type of $[]$, $\mathcal{C}[u]$ denotes the term obtained replacing $[]$ with u in $\mathcal{C}[\ ]$, *without renaming of any bound variable.*

As an example, the expression $\mathcal{C}[\ ] := \lambda x\, z\, ([])$ is a simple context and the term $\lambda x\, z\, (x\, z)$ can be written as $\mathcal{C}[xz]$.

We define below the notion of stack, corresponding to Krivine stack [21] and known as *continuation* because it embodies a series of tasks that wait to be carried out. A stack represents, from the logical perspective, a series of elimination rules; from the λ-calculus perspective, a series of either operations or arguments.

Definition IV.2 (Stack)**.** A **stack** is a sequence $\sigma = \sigma_1\sigma_2 \dots \sigma_n$ such that for every $1 \leq i \leq n$, exactly one of the following holds: either $\sigma_i = t$, with t proof term or $\sigma_i = \pi_j$, with $j \in \{0, 1\}$. We will denote the *empty sequence* with ϵ and with $\xi, \xi', \dots$ the stacks of length 1. If t is a proof term, as usual $t\,\sigma$ denotes the term $(((t\,\sigma_1)\,\sigma_2) \dots \sigma_n)$.

We define now the notion of *strong subformula*, which is essential for defining the reduction rules of the $\mathbb{\lambda}_{\mathrm{G}}$-calculus and for proving Normalization. The technical motivations will become clear in Sections V and VI, but the intuition is that the new types created by cross reductions must be always strong subformulas of already existing types. To define the concept of strong subformula we also need the following definition.

Definition IV.3 (Prime Formulas and Factors [22])**.** 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 IV.4 (Strong Subformula)**.** 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 in the present context, 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 of the strong subformula relation will be often used.

Proposition IV.1 (Characterization of Strong Subformulas)**.** *Suppose B is any strong subformula of A. Then:*

- *If $A = A_1 \wedge \dots \wedge A_n$, with $n > 0$ and $A_1, \dots, A_n$ are prime, then B is a proper subformula of one among $A_1, \dots, A_n$.*
- *If $A = C \to D$, then B is a proper subformula of a prime factor of C or D.*

Proof. Simple considerations on the structure of A. □

Definition IV.5 (Multiple Substitution)**.** Let u be a proof term, $\boldsymbol{x} = x_0^{A_0}, \dots, x_n^{A_n}$ a sequence of variables and $v : A_0 \wedge \dots \wedge A_n$. The substitution $u^{v/\boldsymbol{x}} := u[v\,\pi_0/x_0^{A_0} \dots v\,\pi_n/x_n^{A_n}]$ replaces each variable $x_i^{A_i}$ of any term u with the ith projection of v.

We now seek a measure for determining how complex the communication channel a of a term $u \parallel_a v$ is. Logic will be our guide. First, it makes sense to consider the types B, C such that a occurs with type $B \to C$ in u and thus with type $C \to B$ in v. Moreover, assume $u \parallel_a v$ has type A and its free variables are $x_1^{A_1}, \dots, x_n^{A_n}$. 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 of type more complex than the types of the inputs and the output should appear. Hence, if the prime factors of the types B and C are not subformulas of $A_1, \dots, A_n, A$, then these prime factors should be taken into account in the complexity measure we are looking for. The actual definition is the following.

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

- The pair B, C is called the **communication kind** of a.
- The **communication complexity** of a is the maximum among 0 and the numbers of symbols of the prime factors of B or C that are neither proper subformulas of A nor strong subformulas of one among $A_1, \dots, A_n$.

We explain now the basic reduction rules for the proof terms of $\mathbb{\lambda}_{\mathrm{G}}$, which are given in Figure 1. As usual, we also have the reduction scheme: $\mathcal{E}[t] \mapsto \mathcal{E}[u]$, whenever $t \mapsto u$ and for any context $\mathcal{E}$. With $\mapsto^*$ we shall denote the reflexive and transitive closure of the one-step reduction $\mapsto$.

Intuitionistic Reductions. These are the very familiar computational rules for the simply typed λ-calculus, representing the operations of applying a function and taking a component of a pair [15]. From the logical point of view, they are the standard Prawitz reductions [32] for **NJ**.

Cross Reductions. The reduction rules for (com) model a communication mechanism between parallel processes. In order to apply a cross reduction to a term

$$\mathcal{C}[a\,u] \parallel_a \mathcal{D}[a\,v]$$

several conditions have to be met. These conditions are both natural *and* needed for the termination of computations.
First, we require the communication complexity of a to be greater than 0; again, this is a warning that the Subformula Property does not hold. Here we use a logical property as a *computational criterion* for answering the question: when should computation stop? An answer is crucial here, because, as shown in Example IV.2, unrestricted cross reductions do not always terminate. In λ-calculi the Subformula Property fares pretty well as a stopping criterion. In a sense, it detects all the *essential* operations that really have to be done. For example, in simply typed λ-calculus, a closed term that has the Subformula Property must be a *value*, that is, of the form $\lambda x\,u$, or $\langle u, v\rangle$. Indeed a closed term which is a not a value, must be of the form $h\,\sigma$, for some stack σ (see Definition IV.2), where h is a redex $(\lambda y\,u)t$ or $\langle u, v\rangle\,\pi_i$; but $(\lambda y\,u)$ and $\langle u, v\rangle$ would have a more complex type than the type of the whole term, contradicting the Subformula Property.
Second, we require $\mathcal{C}[a\,u], \mathcal{D}[a\,v]$ to be normal simply typed λ-terms. Simply typed λ-terms, because they are easier to execute in parallel; normal, because we want their computations to go on until they are really stuck and communication is unavoidable.
Third, we require the variable a to be as rightmost as possible and that is needed for logical soundness: how could otherwise the term u be moved to the right, e.g., if it contains a?

Assuming that all the conditions above are satisfied, we can now start to explain the cross reduction

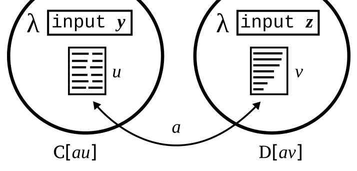
$$\mathcal{C}[a\,u] \parallel_a \mathcal{D}[a\,v] \mapsto (\mathcal{D}[u^{b\langle \boldsymbol{z}\rangle/\boldsymbol{y}}] \parallel_a \mathcal{C}[a\,u]) \parallel_b (\mathcal{C}[v^{b\langle \boldsymbol{y}\rangle/\boldsymbol{z}}] \parallel_a \mathcal{D}[a\,v])$$

Here, the communication channel a has been activated, because the processes $\mathcal{C}$ and $\mathcal{D}$ are synchronized and ready to transfer respectively u and v. The parallel operator $\parallel_a$ let the two occurrences of a communicate: the term u travels to the right in order to replace $a\,v$ and v travels to the left in order to replace $a\,u$. If u and v were data, like numbers or constants, everything would be simple and they could be sent as they are; but in general, this is not possible. The problem is that the free variables $\boldsymbol{y}$ of u which are bound in $\mathcal{C}[a\,u]$ by some λ cannot be permitted to become free; otherwise, the connection between the binders $\lambda\,\boldsymbol{y}$ and the occurrences of the variables $\boldsymbol{y}$ would be lost and they could be no more replaced by actual values when the inputs for the $\lambda\,\boldsymbol{y}$ are available. Symmetrically, the variables $\boldsymbol{z}$ cannot become free. For example, we could have $u = u'\,\boldsymbol{y}$ and $v = v'\,\boldsymbol{z}$ and

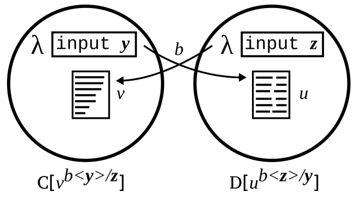
$$\mathcal{C}[a\,u] = w_1\,(\lambda\boldsymbol{y}\,a\,(u'\,\boldsymbol{y})) \qquad \mathcal{D}[a\,v] = w_2\,(\lambda\boldsymbol{z}\,a\,(v'\,\boldsymbol{z}))$$

and the transformation $w_1\,(\lambda\boldsymbol{y}\,a\,(u'\,\boldsymbol{y})) \parallel_a w_2\,(\lambda\boldsymbol{z}\,a\,(v'\,\boldsymbol{z})) \mapsto w_1\,(\lambda\boldsymbol{y}\,v'\,\boldsymbol{z}) \parallel_a w_2\,(\lambda\boldsymbol{z}\,u'\,\boldsymbol{y})$ would just be wrong: the term $v'\,\boldsymbol{z}$ will never get back actual values for the variables $\boldsymbol{z}$ when they will become available.

These issues are typical of *process migration*, and can be solved by the concepts of *code mobility* [14] and *closure* [23]. Informally, code mobility is defined as the capability to dynamically change the bindings between code fragments and the locations where they are executed. Indeed, in order to be executed, a piece of code needs a computational environment and its resources, like data, program counters or global variables. In our case the contexts $\mathcal{C}[\;]$ and $\mathcal{D}[\;]$ are the computational environments or *closures* of the processes u and v and the variables $\boldsymbol{y}, \boldsymbol{z}$ are the resources they need. Now, moving a process outside its environment always requires extreme care: the bindings between a process and the environment resources must be preserved. This is the task of the *migration mechanisms*, which allow a migrating process to resume correctly its execution in the new location. Our migration mechanism creates a new communication channel b between the programs that have been exchanged. Here we see the code fragments u and v, with their original bindings to the global variables $\boldsymbol{y}$ and $\boldsymbol{z}$.



The change of variables $u^{b\langle\boldsymbol{z}\rangle/\boldsymbol{y}}$ and $v^{b\langle\boldsymbol{y}\rangle/\boldsymbol{z}}$ has the effect of reconnecting u and v to their old inputs:



In this way, when they will become available, the data $\boldsymbol{y}$ will be sent to u and the data $\boldsymbol{z}$ will be sent to v through the channel b. Note that in the result of the cross reduction the processes $\mathcal{C}[a\,u]$ and $\mathcal{D}[a\,v]$ are *cloned*, because their code fragments can be needed again. Thus a behaves as a *replicated input* and *replicated output channel*. E.g., in [8], replicated input is coded by the bang operator of linear logic:

$$x\langle y\rangle.Q \mid !x(z).P \mapsto Q \mid P[y/z] \mid !x(z).P$$

With symmetrical message passing and a "!" also in front of $x\langle y\rangle.Q$, one would obtain a version of our cross reduction. Finally, as detailed in Ex. VII.2, whenever u and v are closed terms the cross reduction is simpler and only maintains the first two of the four processes produced in the general case.

Example IV.1 ($\parallel_a$ **in** $\mathbb{\lambda}_{\mathrm{G}}$ **and** $\mid$ **in the** π**-calculus).** A private channel $u \parallel_a v$ is rendered in the π-calculus [26], [33] by the restriction operator ν, as $\nu a\,(u \mid v)$. Recall that the π-calculus term $u \mid v$ represents two processes that run in parallel. The corresponding $\mathbb{\lambda}_{\mathrm{G}}$ term $\langle e, u\rangle \parallel_e \langle e, v\rangle$ is defined using a fresh channel e with communication kind A, A. As no cross reduction outside u and v can be applied, the whole term reduces neither to $\langle e, u\rangle$ nor to $\langle e, v\rangle$, so that u and v can run in parallel.

Example IV.2. Let y and z be bound variables occurring in the normal terms $\mathcal{C}[a\,y]$ and $\mathcal{D}[a\,z]$. Without the condition on the communication complexity c of a, a loop could be generated:

$$\mathcal{C}[a\,y] \parallel_a \mathcal{D}[a\,z] \mapsto (\mathcal{D}[y^{b\langle z\rangle/y}] \parallel_a \mathcal{C}[a\,y]) \parallel_b (\mathcal{C}[z^{b\langle y\rangle/z}] \parallel_a \mathcal{D}[a\,z])$$
$$= (\mathcal{D}[b\,z] \parallel_a \mathcal{C}[a\,y]) \parallel_b (\mathcal{C}[b\,y] \parallel_a \mathcal{D}[a\,z]) \mapsto^* \mathcal{D}[b\,z] \parallel_b \mathcal{C}[b\,y]$$

In Sec. VI we show that if $c > 0$, this reduction sequence would terminate. What is then happening here? Intuitively, $\mathcal{C}[a\,y]$ and $\mathcal{D}[a\,z]$ are normal simply typed λ-terms, which forces $c = 0$.

Permutation Reductions. They regulate the interaction between parallel operators and the other computational constructs. The first four reductions are the Prawitz-style permutation rules [32] between parallel operators and eliminations. We also add two other groups of reductions: three permutations between parallel operators and introductions, two permutations between parallel operators themselves. The first group will be needed to rewrite any proof term into a parallel composition of simply typed λ-terms (Proposition V.3). The second group is needed to address the *scope extrusion* issue of private channels [26]. We point out that a parallel operator $\parallel_a$ is allowed to commute with other parallel operators only when it is strictly necessary, that is, when the communication complexity of a is greater than 0 and thus signaling a violation of the Subformula Property.

Example IV.3 (**Scope extrusion (and** π**-calculus)).** As example of scope extrusion, let us consider the term

$$(v \parallel_a \mathcal{C}[b\,a]) \parallel_b w$$

Here the process $\mathcal{C}[b\,a]$ 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 $\nu a(P \mid Q) \mid R \equiv \nu a(P \mid Q \mid R)$, provided that a does not occur in R, condition that can always be satisfied by α-conversion. Gödel logic offers and actually forces a different solution, which is not just permuting w inward but also duplicating it:

$$(v \parallel_a \mathcal{C}[b\,a]) \parallel_b w \mapsto (v \parallel_b w) \parallel_a (\mathcal{C}[b\,a] \parallel_b w)$$

After this reduction $\mathcal{C}[b\,a]$ can send a to w. If a does not occur in v, we have a further simplification step:

$$(v \parallel_b w) \parallel_a (\mathcal{C}[b\,a] \parallel_b w) \mapsto v \parallel_a (\mathcal{C}[b\,a] \parallel_b w)$$

obtaining associativity of composition as in π-calculus. However, if b occurs in v, this last reduction step is not possible and we keep both copies of w. It is indeed natural to allow both v and $\mathcal{C}[b\,a]$ to communicate with w.

Everything works as expected: the reductions steps in Fig. 1 preserve the type at the level of proof terms, i.e., they correspond to logically sound proof transformations. Indeed

Theorem IV.2 (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 theorem for basic reductions: if $t : A$ and $t \mapsto u$, then $u : A$. The proof that the intuitionistic reductions and the permutation rules preserve the type is completely standard. Cross reductions require straightforward considerations as well. Indeed suppose

$$\mathcal{C}[a^{A\to B}\, u] \parallel_a \mathcal{D}[a^{B\to A}\, v]$$
$$\mapsto$$
$$(\mathcal{D}[u^{b^{D\to C}\langle \boldsymbol{z}\rangle/\boldsymbol{y}}] \parallel_a \mathcal{C}[a^{A\to B}\, u]) \parallel_b (\mathcal{C}[v^{b^{C\to D}\langle \boldsymbol{y}\rangle/\boldsymbol{z}}] \parallel_a \mathcal{D}[a^{B\to A}\, v])$$

Since $\langle \boldsymbol{y}\rangle : C := C_0 \wedge \ldots \wedge C_n$ and $\langle \boldsymbol{z}\rangle : D := D_0 \wedge \ldots \wedge D_m$, $b^{D\to C}\langle \boldsymbol{z}\rangle$ and $b^{C\to D}\langle \boldsymbol{y}\rangle$ are correct terms. Therefore $u^{b^{D\to C}\langle \boldsymbol{z}\rangle/\boldsymbol{y}}$ and $v^{b^{C\to D}\langle \boldsymbol{y}\rangle/\boldsymbol{z}}$, by Definition IV.5, are correct as well. The assumptions are that $\boldsymbol{y} = y_0^{C_0}, \ldots, y_n^{C_n}$ is the sequence of the free variables of u which are bound in $\mathcal{C}[a^{A\to B}u]$, $\boldsymbol{z} = z_0^{D_0}, \ldots, z_m^{D_m}$ is the sequence of the free variables of v which are bound in $\mathcal{D}[a^{B\to A}v]$, a does not occur neither in u nor in v and b is fresh. Therefore, by construction all the variables $\boldsymbol{z}$ are bound in $\mathcal{D}[u^{b^{D\to C}\langle \boldsymbol{z}\rangle/\boldsymbol{y}}]$ and all the variables $\boldsymbol{y}$ are bound in $\mathcal{C}[v^{b^{C\to D}\langle \boldsymbol{y}\rangle/\boldsymbol{z}}]$. Hence, no new free variable is created. □

Definition IV.7 (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_{G}-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_{G} is **normalizable** if there is a finite reduction of u whose last term is a normal form.

Definition IV.8 (Parallel Form). A term t is a **parallel form** whenever, removing the parentheses, it can be written as

$$t = t_1 \parallel_{a_1} t_2 \parallel_{a_2} \ldots \parallel_{a_n} t_{n+1}$$

where each t_i, for $1 \le i \le n+1$, is a simply typed λ-term.

V. The Subformula Property

We show that normal λ_{G}-terms satisfy the important Subformula Property (Theorem V.4). This, in turn, implies that our Curry–Howard correspondence for λ_{G} is meaningful from the logical perspective and produces analytic **NG** proofs.

We start by establishing an elementary property of simply typed λ-terms, which will turn out to be crucial for our normalization proof. It ensures that every bound hypothesis appearing in a normal intuitionistic proof is a strong subformula of one the premises or a proper subformula of the conclusion. This property sheds light on the complexity of cross reductions, because it implies that the new formulas introduced by these operations are always smaller than the local premises.

Proposition V.1 (Bound Hypothesis Property). *Suppose*

$$x_1^{A_1}, \ldots, x_n^{A_n} \vdash t : A$$

$t \in \mathsf{NF}$ is a simply typed λ-term and $z : B$ a variable occurring bound in t. Then one of the following holds:

1) *B is a proper subformula of a prime factor of A.*
2) *B is a strong subformula of one among $A_1, \ldots, A_n$.*

Proof. By induction on t. □

The next proposition says that each occurrence of any hypothesis of a normal intuitionistic proof must be followed by an elimination rule, whenever the hypothesis is neither $\bot$ nor a subformula of the conclusion nor a proper subformula of some other premise.

Proposition V.2. *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 the formulas $A_1, \ldots, A_n$.*

Proof. Easy structural induction on the term. □

Proposition V.3 (Parallel Normal Form Property). *If $t \in \mathsf{NF}$ is a λ_{G}-term, then it is in parallel form.*

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

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

Theorem V.4 (Subformula Property). *Suppose*

$$x_1^{A_1}, \ldots, x_n^{A_n} \vdash t : A \quad \text{and} \quad t \in \mathsf{NF}. \quad \text{Then}:$$

1) *For each communication variable a occurring bound in t and with communication kind B, C, the prime factors of B and C are proper subformulas of $A_1, \ldots, A_n, A$.*
2) *The type of any subterm of t which is not a bound communication variable is either a subformula or a conjunction of subformulas of the formulas $A_1, \ldots, A_n, A$.*

Proof. We proceed by induction on t. By Proposition V.3 $t = t_1 \parallel_{a_1} t_2 \parallel_{a_2} \ldots \parallel_{a_n} t_{n+1}$ and each t_i, for $1 \le i \le n+1$,

Intuitionistic Reductions $\quad (\lambda x^A\, u)t \mapsto u[t/x^A] \quad$ and $\quad \langle u_0, u_1\rangle\, \pi_i \mapsto u_i,$ for $i = 0, 1$

Permutation Reductions

$$(u \parallel_a v)w \mapsto uw \parallel_a vw, \text{ if } a \text{ does not occur free in } w$$

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

$$\mathsf{efq}_P(w_1 \parallel_a w_2) \mapsto \mathsf{efq}_P(w_1) \parallel_a \mathsf{efq}_P(w_2)$$

$$(u \parallel_a v)\, \pi_i \mapsto u\, \pi_i \parallel_a v\, \pi_i$$

$$\lambda x^A\, (u \parallel_a v) \mapsto \lambda x^A\, u \parallel_a \lambda x^A\, v$$

$$\langle u \parallel_a v,\, w\rangle \mapsto \langle u, w\rangle \parallel_a \langle v, w\rangle, \text{ if } a \text{ does not occur free in } w$$

$$\langle w,\, u \parallel_a v\rangle \mapsto \langle w, u\rangle \parallel_a \langle w, v\rangle, \text{ if } a \text{ does not occur free in } w$$

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

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

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

$$\mathcal{C}[a^{A\to B}\, u] \parallel_a \mathcal{D}[a^{B\to A}\, v] \;\mapsto\; (\mathcal{D}[u^{b^{C\to D}\langle \boldsymbol{z}\rangle/\boldsymbol{y}}] \parallel_a \mathcal{C}[a^{A\to B}\, u]) \parallel_b (\mathcal{C}[v^{b^{D\to C}\langle \boldsymbol{y}\rangle/\boldsymbol{z}}] \parallel_a \mathcal{D}[a^{B\to A}\, v])$$

where $\mathcal{C}[a\, u], \mathcal{D}[a\, v]$ are normal simply typed λ-terms and $\mathcal{C}, \mathcal{D}$ simple contexts; $\boldsymbol{y}$ is the sequence of the free variables of u which are bound in $\mathcal{C}[a\, u]$; $\boldsymbol{z}$ is the sequence of the free variables of v which are bound in $\mathcal{D}[a\, v]$; C and D are the conjunctions of the types of the variables in $\boldsymbol{z}$ and $\boldsymbol{y}$, respectively; the displayed occurrences of a are the rightmost both in $\mathcal{C}[a\, u]$ and in $\mathcal{D}[a\, v]$; b is fresh; and the communication complexity of a is greater than 0

Fig. 1. Basic Reduction Rules for $\lambda\!\!\lambda_{\mathrm{G}}$

is a simply typed λ-term. We only show the case $t = u_1 \parallel_b u_2$. Let C, D be the communication kind of b, we first show that the communication complexity of b is 0. We reason by contradiction and assume that it is greater than 0. u_1 and u_2 are either simply typed λ-terms or of the form $v \parallel_c w$. The second case is not possible, otherwise a permutation reduction could be applied to $t \in \mathsf{NF}$. Thus u_1 and u_2 are simply typed λ-terms. Since the communication complexity of b is greater than 0, the types $C \to D$ and $D \to C$ are not subformulas of $A_1, \ldots, A_n, A$. By Prop. V.2, every occurrence of $b^{C\to D}$ in u_1 is of the form $b^{C\to D}v$ and every occurrence of $b^{D\to C}$ in u_2 is of the form $b^{D\to C}w$. Hence, we can write

$$u_1 = \mathcal{C}[b^{C\to D}v] \qquad u_2 = \mathcal{D}[b^{D\to C}w]$$

where $\mathcal{C}, \mathcal{D}$ are simple contexts and b is rightmost. Hence a cross reduction of t can be performed, which contradicts the fact that $t \in \mathsf{NF}$. Since we have established that the communication complexity of b is 0, the prime factors of C and D must be proper subformulas of $A_1, \ldots, A_n, A$. Now, by induction hypothesis applied to $u_1 : A$ and $u_2 : A$, for each communication variable $a^{F\to G}$ occurring bound in t, the prime factors of F and G are proper subformulas of the formulas $A_1, \ldots, A_n, A, C \to D, D \to C$ and thus of the formulas $A_1, \ldots, A_n, A$; moreover, the type of any subterm of u_1 or u_2 which is not a communication variable is either a subformula or a conjunction of subformulas of the formulas $A_1, \ldots, A_n, C \to D, D \to C$ and thus of $A_1, \ldots, A_n, A$. $\square$

Remark V.1. Our statement of the Subformula Property is slightly different from the usual one. However the latter can be easily recovered using the communication rule (com_{end}) of Section III and additional reduction rules. As the resulting derivations would be isomorphic but more complicated, we prefer the current statement.

VI. The Normalization Theorem

Our goal is to prove the Normalization Theorem for $\lambda\!\!\lambda_{\mathrm{G}}$: every proof term of $\lambda\!\!\lambda_{\mathrm{G}}$ reduces in a finite number of steps to a normal form. By Subject Reduction, this implies that **NG** proofs normalize. We shall define a reduction strategy for terms of $\lambda\!\!\lambda_{\mathrm{G}}$: a recipe for selecting, in any given term, the subterm to which apply one of our basic reductions. We remark that the permutations between communications have been adopted to simplify the normalization proof, but at the same time, they undermine strong normalization, because they enable silly loops, like in cut-elimination for sequent calculi. Further restrictions of the permutations might be enough to prove strong normalization, but we leave this as an open problem.

The idea behind our 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.

Definition VI.1 (Complexity of Parallel Terms)**.** Let $\mathcal{A}$ be a finite set of formulas. The $\mathcal{A}$**-complexity** of the term $u \parallel_a v$ is the sequence (c, d, l, o) of natural numbers, where:

1) if the communication kind of a is B, C, then c is the maximum among 0 and the number of symbols of the

prime factors of B or C that are not subformulas of some formula in $\mathcal{A}$;
2) d is the number of occurrences of $\|$ in u and v;
3) l is the sum of the lengths of the intuitionistic reductions of u and v to reach intuitionistic normal form;
4) o is the number of occurrences of a in u and v.

For clarity, we define the recursive normalization algorithm that represents the constructive content of the proofs of Prop. VI.1 and VI.2, which are used to prove the Normalization Theorem. Essentially, our master reduction strategy consists in iterating the basic reduction relation $\succ$ defined below, whose goal is to permute the smallest redex $u \parallel_a v$ of maximal complexity until u and v are simply typed λ-terms, then normalize them and finally apply the cross reductions.

Definition VI.2 (Side Reduction Strategy)**.** Let $t : A$ be a term with free variables $x_1^{A_1}, \dots, x_n^{A_n}$ and $\mathcal{A}$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1, \dots, A_n$. Let $u \parallel_a v$ the *smallest subterm* of t, if any, among those of *maximal* $\mathcal{A}$-complexity and let (c, d, l, o) its $\mathcal{A}$-complexity. We write

$$t \succ t'$$

whenever t' has been obtained from t by applying to $u \parallel_a v$:

1) a permutation reduction

$$(u_1 \parallel_b u_2) \parallel_a v \mapsto (u_1 \parallel_a v) \parallel_b (u_2 \parallel_a v)$$

$$u \parallel_a (v_1 \parallel_b v_2) \mapsto (u \parallel_a v_1) \parallel_b (u \parallel_a v_2)$$

if $d > 0$ and $u = u_1 \parallel_b u_2$ or $v = v_1 \parallel_b v_2$;
2) a sequence of intuitionistic reductions normalizing both u and v, if $d = 0$ and $l > 0$;
3) a cross reduction if $d = l = 0$ and $c > 0$, immediately followed by intuitionistic reductions normalizing the newly generated simply typed λ-terms and, if possible, by applications of the cross reductions $u_1 \parallel_b v_1 \mapsto u_1$ and $u_1 \parallel_b v_1 \mapsto v_1$ to the whole term.
4) a cross reduction $u \parallel_a v \mapsto u$ and $u \parallel_a v \mapsto v$ if $d = l = c = 0$.

Definition VI.3 (Master Reduction Strategy)**.** We define a normalization algorithm $\mathcal{N}(t)$ taking as input a typed term t and producing a term t' such that $t \mapsto^* t'$. The algorithm performs the following operations.

1) If t is not in parallel form, then, using permutation reductions, t is reduced to a t' which is in parallel form and $\mathcal{N}(t')$ is recursively executed.
2) If t is in parallel form, a sequence of terms is produced

$$t \succ t_1 \succ t_2 \succ \ldots \succ t_n$$

such that t_n is not a redex.
3) If t_n is a simply typed λ-term, it is normalized and returned. If $t_n = u \parallel_a v$, then let $\mathcal{N}(u) = u'$ and $\mathcal{N}(v) = v'$. If $u' \parallel_a v'$ is normal, it is returned. Otherwise, $\mathcal{N}(u' \parallel_a v')$ is recursively executed.

We observe that in the step 3 of the algorithm $\mathcal{N}$, by construction $u \parallel_a v$ is not a redex. After u and v are normalized respectively to u' and v', it can still be the case that $u' \parallel_a v'$ is not normal, because some free variables of u and v may disappear during the normalization, causing a new violation of the Subformula Property that transforms $u' \parallel_a v'$ into a redex, even though $u \parallel_a v$ was not.

The first step of the normalization algorithm $\mathcal{N}$ consists in showing that any term can be reduced to a parallel form.

Proposition VI.1. *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 with the help of the algorithm $\mathcal{N}$.

Proposition VI.2. *Let $t : A$ be any term in parallel form. Then $t \mapsto^* t'$, where t' is a normal parallel form.*

Proof. Assume that the free variables of t are $x_1^{A_1}, \dots, x_n^{A_n}$ and let $\mathcal{A}$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1, \dots, A_n$. We prove the theorem by lexicographic induction on the triple

$$(|\mathcal{A}|, (k, r), s)$$

where (k, r) is in turn lexicographically ordered, $|\mathcal{A}|$ is the cardinality of $\mathcal{A}$, k is the number of subterms of t having maximal $\mathcal{A}$-complexity r and s is the size of t. If t is a simply typed λ-term, it has a normal form [15] and we are done; so we assume t is not. There are two main cases.

First case: t *is not a redex*. Let $t = u \parallel_a v$ and let B, C be the communication kind of a. Then, the communication complexity of a is 0 and by Def. IV.6 every prime factor of B or C belongs to $\mathcal{A}$. Let $\mathcal{A}'$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1, \dots, A_n, B \to C$; let $\mathcal{A}''$ be the set of the proper subformulas of A and the strong subformulas of the formulas $A_1, \dots, A_n, C \to B$. By Prop. IV.1, every strong subformula of $B \to C$ or $C \to B$ is a proper subformula of a prime factor of B or C, and this prime factor is in $\mathcal{A}$. Hence, $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{A}'' \subseteq \mathcal{A}$.

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

If $\mathcal{A}' \subset \mathcal{A}$, again by induction hypothesis $u \mapsto^* u'$, where u' is a normal parallel form. The very same argument shows that $v \mapsto^* v'$, where v' is a normal parallel form.

Let now $t' = u' \parallel_a v'$, so that $t \mapsto^* t'$. If t' is normal, we are done. If t' is not normal, since u' and v' are normal, the only possible redex remaining in t' is the whole term itself, i.e., $u' \parallel_a v'$: that 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 $\mathcal{B}$ be the set of the proper subformulas of A

and the strong subformulas of the formulas $A_1, \ldots, A_i$. Since t' is a redex, the communication complexity of a is greater than 0; by Definition IV.6, a prime factor of B or C is not in $\mathcal{B}$, so we have $\mathcal{B} \subset \mathcal{A}$. By induction hypothesis, $t' \mapsto^* t''$, where t'' is a parallel normal form, QED.

Second case: t *is a redex*. We first show that $t \succ t'$, for a t' satisfying Definition VI.2. Let $u \parallel_a v$ be the *smallest* subterm of t having $\mathcal{A}$-complexity r. Four cases can occur.

(a) $r = (c, d, l, o)$, with $d > 0$. First, we prove that $u \parallel_a v$ is a redex showing that the communication complexity of a is greater than 0. Assume that the free variables of $u \parallel_a v$ are among $x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{B_1 \to C_1}, \ldots, a_m^{B_m \to C_m}$ and that the communication kind of a is C, D. Suppose by contradiction that all the prime factors of C and D are proper subformulas of A or strong subformulas of one among $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_m \to C_m$. Given that $c > 0$ there is a prime factor P of C or D such that $P \notin \mathcal{A}$; thus P is a strong subformula of some formula $B_i \to C_i$ and, by Proposition IV.1, a proper subformula of a prime factor of B_i or C_i. Since by hypothesis a_i is bound in t, we conclude that there is a subterm $w_1 \parallel_{a_i} w_2$ of t having $\mathcal{A}$-complexity greater than r, which is absurd.

Now, since $d > 0$, we may assume $u = w_1 \parallel_b w_2$ (the case $v = w_1 \parallel_b w_2$ is symmetric). The term

$$(w_1 \parallel_b w_2) \parallel_a v$$

is then a redex of t and by replacing it with

$$(w_1 \parallel_a v) \parallel_b (w_2 \parallel_a v) \tag{1}$$

we obtain from t a term t' such that $t \succ t'$ according to Def. VI.2. 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 $\mathcal{A}$-complexities are smaller than r, because $u \parallel_a v$ by choice is the smallest subterm of t having maximal $\mathcal{A}$-complexity r. Moreover, the two terms $w_1 \parallel_a v$ and $w_2 \parallel_a v$ have smaller $\mathcal{A}$-complexity than r, because they have numbers of occurrences of the symbol $\parallel$ strictly smaller than in $u \parallel_a v$. Finally, assuming that the communication kind of b is F, G, the prime factors of F and G that are not in $\mathcal{A}$ must have fewer symbols than the prime factors of C and D that are not in $\mathcal{A}$, again because $u \parallel_a v$ by choice is the smallest subterm of t having maximal $\mathcal{A}$-complexity r; hence, the $\mathcal{A}$-complexity of (1) is smaller than r. By induction hypothesis, $t' \mapsto^* t''$, where t'' is a normal parallel form and we are done.

(b) $r = (c, d, l, o)$, with $d = 0$ and $l > 0$. Since $d = 0$, u and v are simply typed λ-terms – and thus strongly normalizable [15] – so we may assume $u \mapsto^* u' \in \mathsf{NF}$ and $v \mapsto^* v' \in \mathsf{NF}$ by a sequence intuitionistic reduction rules. By replacing in t the subterm $u \parallel_a v$ with $u' \parallel_a v'$, we obtain a term t' such that $t \succ t'$ according to Definition VI.2. By induction hypothesis, $t' \mapsto^* t''$, where t'' is a normal parallel form and we are done.

(c) $r = (c, d, l, o)$, with $d = l = 0$ and $c > 0$. Since $d = 0$, u and v are simply typed λ-terms. Since $l = 0$, u and v are in normal form and thus satisfy conditions 1. and 2. of Proposition V.1. We need to check that $u \parallel_a v$ is a redex, in particular that the communication complexity of a is greater than 0. Assume that the free variables of $u \parallel_a v$ are among $x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{B_1 \to C_1}, \ldots, a_m^{B_m \to C_m}$ and that the communication kind of a is C, D. As we argued above, we obtain that not all the prime factors of C and D are proper subformulas of A or strong subformulas of one among $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_m \to C_m$. By Definition IV.6, that is what we wanted to show.

We now prove that every occurrence of a in u and v is of the form $a\,\xi$ for some term or projection ξ. First of all, a occurs with arrow type both in u and v. Moreover, $u : A$ and $v : A$, since $t : A$ and t is a parallel form; hence, the types $C \to D$ and $D \to C$ cannot be subformulas of A, otherwise $c = 0$, and cannot be proper subformulas of one among $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_n \to C_n$, otherwise the prime factors of C, D would be strong subformulas of one among $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_m \to C_m$. Thus by Prop. V.2 we are done. Two cases can occur.

- a does not occur in u or v: to fix ideas, let us say it does not occur in u. By performing a cross reduction, we replace in t the term $u \parallel_a v$ with u and obtain a term t' such that $t \succ t'$ according to Def. VI.2. After the replacement, the number of subterms having maximal $\mathcal{A}$-complexity r in t' is strictly smaller than the number of such subterms in t. By induction hypothesis, $t' \mapsto^* t''$, where t'' is a normal parallel form and we are done.
- a occurs in u and in v. Let $u = \mathcal{C}[a\,w_1\,\sigma]$ and $v = \mathcal{D}[a\,w_2\,\rho]$ where the displayed occurrences of a are the rightmost in u and v and σ, τ are the stacks of *all* terms or projections a is applied to. By applying a cross reduction to $\mathcal{C}[a\,w_1\,\sigma] \parallel_a \mathcal{D}[a\,w_2\,\rho]$ we obtain the term $(*)$

$$(\mathcal{D}[w_1^{b\langle \boldsymbol{z}\rangle/\boldsymbol{y}}\,\rho] \parallel_a \mathcal{C}[a\,w_1]) \parallel_b (\mathcal{C}[w_2^{b\langle \boldsymbol{y}\rangle/\boldsymbol{z}}\,\sigma] \parallel_a \mathcal{D}[a\,w_2])$$

By hypothesis, $\boldsymbol{y}$ is the sequence of the free variables of w_1 which are bound in $\mathcal{C}[a\,w_1\,\sigma]$ and $\boldsymbol{z}$ is the sequence of the free variables of w_2 which are bound in $\mathcal{D}[a\,w_2\,\rho]$ and a does not occur neither in w_1 nor in w_2. Since u, v satisfy conditions 1. and 2. of Proposition V.1 the types $Y_1, \ldots, Y_i$ and $Z_1, \ldots, Z_j$ of respectively the variables $\boldsymbol{y}$ and $\boldsymbol{z}$ are proper subformulas of A or strong subformulas of the formulas $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_m \to C_m$. Hence, the types among $Y_1, \ldots, Y_i, Z_1, \ldots, Z_j$ which are not in $\mathcal{A}$ are strictly smaller than all the prime factors of the formulas $B_1, C_1, \ldots, B_m, C_m$. Since the communication kind of b is $Y_1 \wedge \ldots \wedge Y_i, Z_1 \wedge \ldots \wedge Z_j$, by Definition VI.1 either the $\mathcal{A}$-complexity of the term $(*)$ above is strictly smaller than the $\mathcal{A}$-complexity r of $u \parallel_a v$, or the communication kind of b is $\top$. In the latter case we apply a cross reduction $u_1 \parallel_b v_1 \mapsto u_1$ or $u_1 \parallel_b v_1 \mapsto v_1$ and obtain a term with $\mathcal{A}$-complexity strictly smaller than r.

In the former case, let w_1', w_2' be simply typed λ-terms such that

$$w_1^{b\langle \boldsymbol{z}\rangle/\boldsymbol{y}}\,\rho \mapsto^* w_1' \in \mathsf{NF} \text{ and } w_2^{b\langle \boldsymbol{y}\rangle/\boldsymbol{z}}\,\sigma \mapsto^* w_2' \in \mathsf{NF}$$

By hypothesis, a does not occur in w_1, w_2, σ, ρ and thus neither in w_1' nor in w_2'. Moreover, by the assumptions on σ and ρ and since $\mathcal{C}[a\, w_1\, \sigma]$ and $\mathcal{D}[a\, w_2\, \rho]$ are normal simply typed λ-terms, $\mathcal{C}[w_2']$ and $\mathcal{D}[w_1']$ are normal too and contain respectively one fewer occurrence of a than the former terms. Hence, the $\mathcal{A}$-complexity of the terms

$$\mathcal{D}[w_1'] \parallel_a \mathcal{C}[a\, w_1] \quad \text{and} \quad \mathcal{C}[w_2'] \parallel_a \mathcal{D}[a\, w_2]$$

is strictly smaller than the $\mathcal{A}$-complexity r of $u \parallel_a v$. Let now t' be the term obtained from t by replacing the term $\mathcal{C}[a\, w_1\, \sigma] \parallel_a \mathcal{D}[a\, w_2\, \rho]$ with

$$(\mathcal{D}[w_1'] \parallel_a \mathcal{C}[a\, w_1]) \parallel_b (\mathcal{C}[w_2'] \parallel_a \mathcal{D}[a\, w_2])$$

By construction $t \succ t'$. Hence, we can apply the main induction hypothesis to t' and obtain $t' \mapsto^* t''$, where t'' is a normal parallel form and we are done.

(d) $r = (c, d, l, o)$, with $d = l = c = 0$. Since t is a redex, then $t = u_1 \parallel_b v_1$ where b does not occur in u_1 or v_1: to fix ideas, let us say it does not occur in u_1. By performing a cross reduction, we replace t with u_1 so that $t \succ u_1$ according to Def. VI.2. Hence, we can apply the main induction hypothesis to u_1 and obtain $u_1 \mapsto^* t''$, where t'' is a normal parallel form and we are done. □

The normalization for λ_{G}, and thus for **NG**, easily follows.

Theorem VI.3. *Suppose that* $t : A$ *is a proof term of* **G**. *Then* $t \mapsto^* t' : A$, *where* t' *is a normal parallel form.*

VII. Computing with λ_{G}

We illustrate the expressive power of λ_{G} by a few examples. All the examples employ the normalization algorithm in Definition VI.3; to limit its non-determinism, when we have to reduce $u \parallel_a v$ because a does not occur neither in u nor in v, we always use the reduction $u \parallel_a v \mapsto u$.

Henceforth we use the types $\mathbb{N}$ for natural numbers, Bool for the Boolean values and String for strings.

We start by showing that λ_{G} is more expressive than simply typed λ-calculus.

Example VII.1 (**Parallel or**). Berry's sequentiality theorem (see [15]) implies that there is no λ-term $\mathsf{O} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ such that $\mathsf{OFF} \mapsto \mathsf{F}$, $\mathsf{O}u\mathsf{T} \mapsto \mathsf{T}$, $\mathsf{OT}u \mapsto \mathsf{T}$, where u is an arbitrary normal term, and thus possibly a variable. O can instead be defined in Boudol's parallel λ-calculus [7].

The λ_{G} term for such parallel or is (as usual the term "$\mathsf{if}\, u\, \mathsf{then}\, s\, \mathsf{else}\, t$" reduces to s if $u = \mathsf{T}$, and to t if $u = \mathsf{F}$):

$$\begin{aligned}\mathsf{O} := \lambda x^{\mathsf{Bool}}\, \lambda y^{\mathsf{Bool}}\, &(\mathsf{if}\, x\, \mathsf{then}\, (\lambda z\, \lambda k\, z)\, \mathsf{else}\, (\lambda z\, \lambda k\, k))\mathsf{T}(ax) \\ &\parallel_a (\mathsf{if}\, y\, \mathsf{then}\, (\lambda z\, \lambda k\, z)\, \mathsf{else}\, (\lambda z\, \lambda k\, k))\mathsf{T}(ay)\end{aligned}$$

where the communication kind of a is $\mathsf{Bool}, \mathsf{Bool} \wedge \mathbb{N}$. Now

$$\begin{aligned}\mathsf{O}\, u\, \mathsf{T} &\mapsto^* (\mathsf{if}\, u\, \mathsf{then}\, (\lambda z\, \lambda k\, z)\, \mathsf{else}\, (\lambda z\, \lambda k\, k))\mathsf{T}(au) \\ &\qquad \parallel_a (\mathsf{if}\, \mathsf{T}\, \mathsf{then}\, (\lambda z\, \lambda k\, z)\, \mathsf{else}\, (\lambda z\, \lambda k\, k))\mathsf{T}(a\mathsf{T}) \\ &\mapsto^* (\mathsf{if}\, u\, \mathsf{then}\, (\lambda z\, \lambda k\, z)\, \mathsf{else}\, (\lambda z\, \lambda k\, k))\mathsf{T}(au) \parallel_a \mathsf{T} \mapsto \mathsf{T}\end{aligned}$$

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

$$\begin{aligned}\mathsf{O\,F\,F} \mapsto^* &\quad (\lambda z\, \lambda k\, k)\mathsf{T}(a\mathsf{F}) \parallel_a (\lambda z\, \lambda k\, k)\mathsf{T}(a\mathsf{F}) \\ \mapsto^* &\quad (a\mathsf{F}) \parallel_a (a\mathsf{F}) \\ \mapsto^* &\quad (\mathsf{F} \parallel_a (a\mathsf{F})) \parallel_b (\mathsf{F} \parallel_a (a\mathsf{F})) \quad \mapsto^* \quad \mathsf{F}\end{aligned}$$

Example VII.2 (**Data passing**). As in the previous example, if the messages sent during a cross reduction are closed terms, for example data, the outcome is a simple unidirectional message passing. Indeed, the newly introduced communication is void and is always removed:

$$\mathcal{C}[a\, u] \parallel_a \mathcal{D}[a\, v] \mapsto$$

$$(\mathcal{D}[u] \parallel_a \mathcal{C}[a\, u]) \parallel_b (\mathcal{C}[v] \parallel_a \mathcal{D}[a\, v]) \mapsto \mathcal{D}[u] \parallel_a \mathcal{C}[a\, u]$$

If we want a process s to transmit a message $m : B$ to a process t without t passing anything back, we can use the following term (a has communication kind $(B \to F) \to F, F \to F$):

$$(a\lambda z^{A\to F}\, zm)s \parallel_a (a\lambda y^{F}\, y)(\lambda x^{B}\, t) \mapsto$$

$$((\lambda z\, zm)(\lambda x\, t) \parallel_a (a\lambda z\, zm)s) \parallel_e ((\lambda y\, y)s \parallel_a (a\lambda y\, y)(\lambda x\, t))$$

$$\mapsto^* \quad (\lambda z\, zm)(\lambda x\, t) \parallel_e (\lambda y\, y)s \quad \mapsto^* \quad t[m/x] \parallel_e s$$

This reduction resembles indeed the unidirectional communication $\overline{a}\langle m\rangle.P \mid a(x).Q \mapsto P \mid Q[m/x]$ in the π-calculus [26], [33], assuming a does not occur in P and Q.

In the following example, similar to that in [8], we simulates the communication needed to conclude an online sale.

Example VII.3 (**Buyer and vendor**). We model the following transaction: a buyer tells a vendor a product name $\mathsf{prod} : \mathsf{String}$, the vendor computes the value $\mathsf{price} : \mathbb{N}$ of prod and sends it to the buyer, the buyer sends back the credit card number $\mathsf{card} : \mathsf{String}$ which is used to pay.

We introduce the following functions: $\mathsf{cost} : \mathsf{String} \to \mathbb{N}$ with input a product name prod and output its cost price; $\mathsf{pay\_for} : \mathbb{N} \to \mathsf{String}$ with input a price and output a credit card number card; $\mathsf{use} : \mathsf{String} \to \mathbb{N}$ that obtains money using as input a credit card number $\mathsf{card} : \mathsf{String}$. The buyer and the vendor are the contexts $\mathcal{B}$ and $\mathcal{V}$ of type Bool. Notice that the terms representing buyer and vendor exchange their position at each cross reduction. For a of kind $\mathsf{String}, \mathbb{N}$, the program is:

$$\begin{aligned}&\mathcal{B}[a(\mathsf{pay\_for}(a(\mathsf{prod})))] \parallel_a \mathcal{V}[\mathsf{use}(a(\mathsf{cost}(a\, 0)))] \\ &\mapsto^* \mathcal{V}[\mathsf{use}(a(\mathsf{cost}(\mathsf{prod})))] \parallel_a \mathcal{B}[a(\mathsf{pay\_for}(a(\mathsf{prod})))] \\ &\mapsto \mathcal{V}[\mathsf{use}(a(\mathsf{price}))] \parallel_a \mathcal{B}[a(\mathsf{pay\_for}(a(\mathsf{prod})))] \\ &\mapsto^* \mathcal{B}[a(\mathsf{pay\_for}(\mathsf{price}))] \parallel_a \mathcal{V}[\mathsf{use}(a(\mathsf{price}))] \\ &\mapsto \mathcal{B}[a(\mathsf{card})] \parallel_a \mathcal{V}[\mathsf{use}(a(\mathsf{price}))] \mapsto^* \mathcal{V}[\mathsf{use}(\mathsf{card})] \parallel_a \mathcal{B}[a(\mathsf{card})]\end{aligned}$$

Finally $\mapsto \mathcal{V}[\mathsf{use}(\mathsf{card})]$: the buyer has performed its duty and the vendor uses the card number to obtain the due payment.

We show that although more complicated than sending data, sending open processes can enhance efficiency.

Example VII.4 (Efficiency via cross reductions). Given three processes $M \parallel_d (P \parallel_a Q)$. Assume that Q wants to send a process to P, but one of the process' parameters is not available because M first needs many time-consuming steps to produce it and only afterwards can send it to Q. Cross reductions make it possible to fully exploit parallelism and improve the program efficiency: Q does not need to wait that much and can send the process directly to P, which can begin to partially evaluate it with no further delay. After having computed the data, M sends it to Q which in turn forwards it to P.

For a concrete example, assume that

$$\begin{array}{rcl} M & \mapsto^* & d\,(\lambda k^{\mathbb{N}\to\mathbb{N}\to\mathbb{N}}\, k\,7\,0) \\ P & = & d\,0\,(\lambda j^{\mathbb{N}}\,\lambda x^{\mathbb{N}}\,(ax)5s) \\ Q & = & d\,0\,(\lambda y^{\mathbb{N}}\,\lambda l^{\mathbb{N}}\,a(\lambda z^{\mathbb{N}}\,\lambda i^{\sigma}\,h\langle g(z), y\rangle)) \end{array}$$

where $h : \mathbb{N} \wedge \mathbb{N} \to \mathbb{N}$, $g : \mathbb{N} \to \mathbb{N}$, the communication kind of d is $(\mathbb{N} \to \mathbb{N} \to \mathbb{N}) \to \mathbb{N}, \mathbb{N}$, and the communication kind of a is $\mathbb{N}, \mathbb{N} \to \sigma \to \mathbb{N}$ with σ arbitrary type of high complexity. Here Q wants to send $\lambda z^{\mathbb{N}}\,\lambda i^{\sigma}\,h\langle g(z), y\rangle$ to P, but the value 7 of the parameter y is computed and transmitted to Q by M only later. On the other hand, P waits for the process from Q in order to instantiate z with 5 and compute $h\langle g(5), 7\rangle$.

Without a special mechanism for sending open terms, P must wait for M to normalize. Afterwards M passes $(\lambda k\, k\,7\,0)$ through d to P and Q with the following computation:

$$\begin{array}{l} M \parallel_d (P \parallel_a Q) \mapsto^* (\lambda k\, k\,7\,0)(\lambda j\,\lambda x\,(ax)5s) \parallel_a \\ \qquad\qquad\qquad (\lambda k\, k\,7\,0)(\lambda y\,\lambda l\, a(\lambda z\,\lambda i\, h\langle g(z), y\rangle)) \mapsto^* \\ (a\,0)5s \parallel_a a(\lambda z\,\lambda i\, h\langle g(z), 7\rangle) \mapsto^* (\lambda z\,\lambda i\, h\langle g(z), 7\rangle)5s \mapsto^* h\langle g(5), 7\rangle \end{array}$$

Our normalization algorithm allows instead Q to directly send $\lambda z^{\mathbb{N}}\,\lambda i^{\sigma}\,h\langle g(z), y\rangle$ to P by executing first a cross reduction:

$$\begin{array}{l} M \parallel_d \Big(d\,0(\lambda j\,\lambda x\,(ax)5s) \parallel_a d\,0(\lambda y\,\lambda l\, a(\lambda z\,\lambda i\, h\langle g(z), y\rangle))\Big) \\ \mapsto M \parallel_d \Big((d\,0(\lambda y\,\lambda l\, by)\|_a P) \parallel_b \big(d\,0(\lambda j\,\lambda x\,(\lambda z\,\lambda i\, h\langle g(z), bx\rangle)5s) \parallel_a Q\big)\Big) \\ \mapsto^* M \parallel_d \Big(d\,0(\lambda y\,\lambda l\, by) \parallel_b d\,0(\lambda j\,\lambda x\,(\lambda z\,\lambda i\, h\langle g(z), bx\rangle)5s)\Big) \end{array}$$

where the communication b (of kind $\mathbb{N}, \mathbb{N}$) redirects the data x and y. Then P instantiates z with 5 and can compute for example $g(5) = 9$ without having to evaluate $h\langle g(5), 7\rangle$ all at once. When M terminates the computation, sends 7 to the new location of the partially evaluated processes P and Q via $\parallel_b$:

$$\begin{array}{l} \mapsto^* M \parallel_d \Big(d\,0(\lambda y\,\lambda l\, by) \parallel_b d\,0(\lambda j\,\lambda x\, h\langle g(5), bx\rangle)\Big) \\ \mapsto^* \Big(M \parallel_d d\,0(\lambda y\,\lambda l\, by)\Big) \parallel_b \Big(M \parallel_d d\,0(\lambda j\,\lambda x\, h\langle 9, bx\rangle)\Big) \\ \mapsto^* (\lambda k\, k\,7\,0)(\lambda y\,\lambda l\, by) \parallel_b (\lambda k\, k\,7\,0)(\lambda j\,\lambda x\, h\langle 9, bx\rangle) \\ \mapsto^* \quad b7 \parallel_b h\langle 9, b\,0\rangle \quad \mapsto^* \quad h\langle 9, 7\rangle \end{array}$$

Final Remark The Curry–Howard isomorphism for λ_{G} interprets Gödel logic in terms of communication between parallel processes. In addition to revealing this connection, our results pave the way towards a more general computational interpretation of the intermediate logics formalized by hypersequent calculi. These logics are characterized by disjunctive axioms of a suitable form [9] – containing all the disjunctive tautologies of [11] – and likely correspond to other communication mechanisms between parallel processes.

References

[1] F. Aschieri. On Natural Deduction for Herbrand Constructive Logics I: Curry–Howard Correspondence for Dummett's Logic LC. *Log. Methods Comput. Sci.*, vol. 12(3) n. 13, pp. 1–31, 2016.

[2] F. Aschieri, M. Zorzi. On Natural Deduction in Classical First-Order Logic: Curry–Howard Correspondence, Strong Normalization and Herbrand's Theorem. *Theoret. Comput. Sci.*, 625: 125–146, 2016.

[3] A. Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Ann. Math. Artif. Intell.*, 4: 225–248, 1991.

[4] A. Avron. The method of hypersequents in the proof theory of propositional non-classical logic. In *Logic: From Foundations to Applications*. Oxford University Press, pp. 1–32, 1996.

[5] M. Baaz, A. Ciabattoni, C. Fermüller. A Natural Deduction System for Intuitionistic Fuzzy Logic. In *Lectures on Soft Computing and Fuzzy Logic*, pp. 1–18, Physica-Verlag, 2000.

[6] A. Beckmann and N. Preining. Hyper natural deduction. In *LICS 2015*, pp. 547–558, 2015.

[7] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. *TAPSOFT '98*, pp. 149-161, vol. 1, 1989.

[8] L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR 2010* pages 222-236. LNCS 6269, 2010.

[9] A. Ciabattoni, N. Galatos and K. Terui. From axioms to analytic rules in nonclassical logics. In *LICS 2008*, pp. 229–240, 2008.

[10] A. Ciabattoni and F. A. Genco. Embedding formalisms: hypersequents and two-level systems of rules. In *AIML 2016*, pp. 197–216, 2016.

[11] V. Danos, J.-L. Krivine. Disjunctive Tautologies as Synchronisation Schemes. In *CSL 2000*, 1862: 292–301, 2000.

[12] M. Dummett. A propositional calculus with denumerable matrix. *J. Symbolic Logic*, 24: 97–106, 1959.

[13] D. Flanagan. JavaScript: the Definitive Guide, O'Reilly Media, 2011.

[14] A. Fuggetta, G.P. Picco and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24: 342–361, 1998.

[15] J.-Y. Girard and Y. Lafont and P. Taylor, *Proofs and Types*. Cambridge University Press, 1989.

[16] K. Gödel: Zum intuitionistischen Aussagenkalkül. *Anzeiger der Kaiserlichen Akademie der Wissenschaften, Mathematisch-Naturwissenschaftliche Classe. Wien*. 69: 65–66, 1932.

[17] T. Griffin. A Formulae-as-Type Notion of Control. In *POPL 1990*, 1990.

[18] P. de Groote, A Simple Calculus of Exception Handling, *Proceedings of TLCA 1995*, LNCS, vol. 902 pp. 201–215, 1995.

[19] Y. Hirai. A lambda calculus for Gödel–Dummett logic capturing waitfreedom. In *FLOPS 2012*, pp. 151–165, 2012.

[20] W. A. Howard. 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. 1980.

[21] J.-L. Krivine. Classical Realizability. *Interactive models of computation and program behavior, Panoramas et synthèses*, pp. 197–229, 2009.

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

[23] P. J. Landin. The Mechanical Evaluation of Expressions. *The Computer Journal*, 6(4), pp. 308–320, 1964.

[24] E.G.K. Lopez-Escobar. Implicational logics in natural deduction systems. *J. Symbolic Logic*, 47(1): 184–186, 1982.

[25] G. Metcalfe and N. Olivetti and D. Gabbay. Proof Theory for Fuzzy Logics. *Springer Series in Applied Logic* vol. 36, 2008.

[26] R. Milner. Functions as Processes. *Mathematical Structures in Computer Science*, vol. 2, n. 2, pp. 119–141,1992.

[27] D. Mostrous, N. Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, vol. 241, pp. 227–263, 2015.

[28] T. Murphy, K. Crary, R. Harper, F. Pfenning. A Symmetric Modal Lambda Calculus for Distributed Computing. In *LICS 2004*, pp. 286–295, 2004.

[29] S. Negri. Proof analysis beyond geometric theories: from rule systems to systems of rules. *J. Logic Comput.*, vol. 27, pp. 513-537, 2016.

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

[31] J. A. Pérez. Higher-Order Concurrency: Expressiveness and Decidability Results, PhD thesis, University of Bologna, 2010.

[32] D. Prawitz. Ideas and Results in Proof Theory. In *Proceedings of the Second Scandinavian Logic Symposium*, 1971.

[33] D. Sangiorgi and D. Walker. The pi-calculus: a Theory of Mobile Processes. 2003.

[34] P. Wadler. Propositions as Types. *Communications of the ACM*, 58(12): 75–84, 2015.