Disjunctive Axioms and Concurrent $\lambda$-Calculi: a Curry–Howard Approach*

Federico Aschieri  
TU Wien, Vienna (Austria)

Agata Ciabattoni  
TU Wien, Vienna (Austria)

Francesco A. Genco  
TU Wien, Vienna (Austria)

Abstract

We add to intuitionistic logic infinitely many classical disjunctive tautologies and use the Curry–Howard correspondence to obtain typed concurrent $\lambda$-calculi; each of them features a specific communication mechanism, including broadcasting and cyclic message-exchange, and enhanced expressive power w.r.t. the $\lambda$-calculus. Moreover they all implement forms of code mobility. Our results provide a first concurrent computational interpretation for many propositional intermediate logics, classical logic included.

Keywords  
Natural deduction, Curry–Howard, lambda calculus, intermediate logics, hypersequents

1 Introduction

Although definable in three lines, the $\lambda$-calculus provides an elegant, yet powerful computational theory. Formally introduced by Church, it has been discovered three times, giving rise to a beautiful connection between three different fields. In mathematics, it was informally used as a simple syntax for function definition and $\lambda$-interpretation of linear logic as session-typed processes to extend the $\lambda$-calculus is the $\pi$-calculus [22, 28] – the most wide-ranging extension of the $\lambda$-calculus have been defined, for instance, by $\mu$-calculus [24]. It arises by interpreting via the Curry–Howard correspondence Pierce’s axiom $(A \to B) \to A \to B$ of classical logic, first used in [13] to provide a type for the call/cc operator of Scheme and in the $\lambda C$-calculus [15]. Particularly challenging is to extend the $\lambda$-calculus with mechanisms for message exchanges; the latter are at the core of the $\pi$-calculus [22, 28] – the most widespread formalism for modeling concurrent systems. Concurrent extensions of the $\lambda$-calculus have been defined, for instance, by adding communication mechanisms inspired by process calculi (e.g. [5]) or by programming languages (e.g. [23]). Logic–based extensions are mainly rooted in variants and extensions of linear logic. Indeed the functional languages in [27, 30] use the Curry–Howard interpretation of linear logic as session-typed processes to extend the $\lambda$-calculus (linear, for [30] with primitives for $\pi$-calculus-based session-typed communication [6]. Enhancing the base logics with suitable axioms allow additional features and programming language constructs to be captured, see, e.g., [7]. A challenge is to find such axioms. One of them was investigated in [1], where a concurrent reading of $(G) = (A \to B) \lor (B \to A)$ was proposed. There, by lifting the original Curry–Howard correspondence in [18] to Gödel logic (intuitionistic logic IL extended with $(G)$) a new typed concurrent $\lambda$-calculus was introduced. $\lambda G$, as it is called, turns out to be an extension of the simply typed $\lambda$-calculus with a parallel operator connecting two processes and a mechanism for symmetric message-exchange. This mechanism is directly extracted from the normalization proof for the natural deduction system for Gödel logic, similarly to how the $\beta$-reduction arises from normalization of intuitionistic derivations. As a result each construct in $\lambda G$ has a counterpart in the logic, making programs isomorphic to logical proofs, as opposed to the calculi in [27, 30].

Disjunctive Tautologies, Hypersequents and Concurrency

In this paper we identify infinitely many axioms, each leading to a different communication mechanism and use the Curry–Howard correspondence to obtain new typed concurrent $\lambda$-calculi. Our axioms belong to the classical tautologies interpreted in [10] as synchronization schemes – $(G)$ being one of them. For these formulas, [10] provides realizers in a concurrent extension of the $\lambda C$-calculus [15], but the question of developing concurrent calculi based on them and Curry–Howard isomorphisms for the corresponding intermediate logics was left open. A further and independent hint on the connection between these formulas and concurrency comes from proof theory: they all belong to the class $P_2$ of the classification in [8] and can therefore be transformed into equivalent structural rules in the hypersequent calculus. Avron – who introduced this framework – suggested in [2] that IL extended with axioms equivalent to such rules could serve as bases for concurrent $\lambda$-calculi. Hypersequents are indeed nothing but parallel sequents whose interaction is governed by special rules that allow the sequents to “communicate”, e.g., [3]. Different rules enable different types of communications between sequents. Despite their built in “concurrency” and various attempts (e.g. [4, 17]), hypersequent rules do not seem to be good bases for a Curry–Howard interpretation, and Avron’s claim has remained unsolved for more than 25 years. What we need are instead natural deduction inferences that, if well designed, can be interpreted as program instructions, and in particular as typed $\lambda$-terms. Normalization [25], which corresponds to the execution of the resulting programs, can then be used to obtain analytic proofs, i.e. proofs containing only formulas that are subformulas of hypotheses or conclusion. Attempts to define such natural deduction inferences capturing some of the considered axioms are, e.g., in [4, 16]; nevertheless, in these texts normalization is missing or the introduced rules “mirror” the hypersequent structure, thus hindering their computational reading. For the particular case of Gödel logic a well-designed calculus...
was introduced in [1] by extending the natural deduction system NJ for IL with an inference rule simulating the hypersequent rule for (G). The reduction rules required to obtain analytic proofs led to the definition of a specific typed concurrent functional language.

**Curry–Howard correspondence: the concurrent calculi \( \lambda_{Ax} \)**

In this paper we exploit the intuitions in [2, 10] to define infinitely-many logic-based typed concurrent \( \lambda \)-calculi. The calculi are extracted – using the Curry–Howard correspondence – from IL extended with classical tautologies of the form \( \lor (A_i \rightarrow B_i) \) where \( A_i \) is a conjunction of propositional variables (possibly \( \top \)) and \( B_i \) is a propositional variable or \( \bot \), and the conjunctions in all \( A_i \)'s are distinct. Examples of such formulas include (EM), \( A \lor \neg A \), \( (A_n) (A_1 \rightarrow A_2) \lor \ldots \lor (A_n \rightarrow A_1) \), and \((G_n)(A_1 \rightarrow A_2) \lor \ldots \lor (A_{n-1} \rightarrow A_n) \lor \neg A_n \), for \( n \geq 2 \); their addition to IL leads to classical logic, cyclic IL [16] (Gödel logic, for \( n = 2 \)), and \( n \)-valued Gödel logic, respectively. Let \( Ax \) be any finite set of such axioms. The corresponding calculus \( \lambda_{Ax} \) arises from IL + Ax as the simply typed \( \lambda \)-calculus arises from IL. Examples of our \( \lambda \)-calculus \( \lambda_{Ax} \) are:

\[
\lambda_{EM}^1: \text{the simplest message-passing mechanism à la } \pi-\text{calculus} \\
\lambda_{EM}^n: \text{cyclic communication among } n \text{ processes.}
\]

Notice that \( \lambda_{EM} \) provides a new computational interpretation of classical logic, obtained using \( A \lor \neg A \) in the Curry–Howard correspondence. It was indeed a long-standing open problem to provide an analytic natural deduction based on (EM) and enjoying a significant computation interpretation: see [14] for an attempt.

Furthermore, different syntactic forms of the same axiom lead to different \( \lambda \)-calculi, e.g. for (EM)_n \( A \lor \ldots \lor A \lor \neg A \) we have

\[
\lambda_{EM}^n: \text{broadcast messages to } n \text{ processes.}
\]

The natural deduction calculus used as type system for \( \lambda_{Ax} \) is defined by extending NJ with the natural deduction version [9] of the hypersequent rules for \( Ax \). The decorated version of these rules lead to \( \lambda \)-calculus with parallel operators that bind two or more processes, according to the shape of the rules. For example the decorated version of the rules for (EM) and (C3) are:

\[
\begin{align*}
\Gamma & : A \\
\frac{\Delta \vdash \Gamma \vdash a \text{ : } A}{a \text{ : } A} & \quad \frac{\Delta \vdash \Gamma \vdash u \text{ : } A}{u \text{ : } A} \\
\frac{\Delta \vdash \Gamma \vdash a \text{ : } A}{a \text{ : } A} & \quad \frac{\Delta \vdash \Gamma \vdash u \text{ : } B}{u \text{ : } B} \\
\frac{\Delta \vdash \Gamma \vdash u \text{ : } B}{u \text{ : } B} & \quad \frac{\Delta \vdash \Gamma \vdash \neg \text{ : } C}{\neg \text{ : } C} \\
\frac{\Delta \vdash \Gamma \vdash \neg \text{ : } C}{\neg \text{ : } C} & \quad \frac{\Delta \vdash \Gamma \vdash w \text{ : } C}{w \text{ : } C} \\
\frac{\Delta \vdash \Gamma \vdash w \text{ : } C}{w \text{ : } C} & \quad \frac{\Delta \vdash \Gamma \vdash (\text{ : } A)}{(\text{ : } A)} \\
\frac{\Delta \vdash \Gamma \vdash (\text{ : } A)}{(\text{ : } A)} & \quad \frac{\Delta \vdash \Gamma \vdash t \text{ : } F}{t \text{ : } F} \\
\frac{\Delta \vdash \Gamma \vdash t \text{ : } F}{t \text{ : } F} & \quad \frac{\Delta \vdash \Gamma \vdash s \text{ : } F}{s \text{ : } F} \\
\frac{\Delta \vdash \Gamma \vdash s \text{ : } F}{s \text{ : } F} & \quad \frac{\Delta \vdash \Gamma \vdash r \text{ : } F}{r \text{ : } F} \\
\frac{\Delta \vdash \Gamma \vdash r \text{ : } F}{r \text{ : } F} & \quad \frac{\Delta \vdash \Gamma \vdash d \text{ : } s \parallel t \text{ : } F}{d \text{ : } s \parallel t \text{ : } F} \\
\frac{\Delta \vdash \Gamma \vdash d \text{ : } s \parallel t \text{ : } F}{d \text{ : } s \parallel t \text{ : } F}
\end{align*}
\]

The variable \( a \) represents a private communication channel that behaves similarly to the \( \pi \)-calculus restriction operator \( v \) [22, 28]. Our natural deduction calculi are defined in a modular way and the rules added to NJ are higher-level [26], as they also discharge rule applications rather than only formulas. We provide a normalization proof that uniformly applies to all the associated \( \lambda \)-calculi. Our reduction rules and normalization strategy mark a radical departure from [1]. They are driven not only by logic, but also by programming. Indeed, while in [1] processes can only communicate when the type of their channels violates the subformula property, our new reductions are logic independent and are activated when messages are values (cf. Def. 3.6). As a result, programs are easier to write and and their evaluation exhausts all active sessions. The normalization proof employs sophisticated termination arguments and works for the full logic language, \( \lor \) included. Since it does not exploit properties specific to Gödel logic, it can be used in a general setting. Its instantiation leads to computational reductions that specify the mechanism for message-exchange in the corresponding typed \( \lambda \)-calculus.

For any logic IL + Ax and \( \lambda_{Ax} \) we show the perfect match between computation steps and proof reductions (Subject Reduction), a terminating reduction strategy for \( \lambda_{Ax} \) (Normalization) and the Subformula Property. Our logic-based calculi are more expressive than the \( \lambda \)-calculus, and their typed versions are non-deterministic\(^2\). Moreover they can implement multiparty communication (in contrast with \( \lambda_{CG} \) [1], cf. Sec. 3.3) and, although deadlock-free, communication between cyclic interconnected processes (in contrast with [6], cf. Ex. 4.2). Furthermore the computational reductions associated to each axiom enjoy a natural interpretation in terms of higher-order process passing and implement forms of code mobility [11]; the latter can be used to improve efficiency of programs: open processes can be sent first and new communication channels taking care of their closures can be created afterwards (Ex. 3.2).

The paper is organized as follows: Sec. 2 defines the class of considered logics and their natural deduction systems. In Sec. 3 we use the typed concurrent calculus \( \lambda_{EM} \) for classical logic as a case study to explain the ideas behind the costruction of our calculi, their reductions and the normalization proof. Examples of programs, including broadcasting, are also given. Sec. 4 deals with the general case: the definition of the typed concurrent calculus \( \lambda_{Ax} \) parametrized on the set of axioms \( Ax \). A sophisticated normalization proof working for all of them is presented in Sec. 5.

2 Disjunctive Axioms and Natural Deduction

We introduce natural deductions for infinitely many intermediate logics obtained by extending IL by any set of Hilbert axioms

\[
\bigwedge_{i=1, \ldots, k} \bigvee_{j=1, \ldots, n_i} A_{ij} 
\]

where \( A_{ij} \) and \( B_i \) for \( i = 1, \ldots, k \) is a propositional variable or \( \top \) or \( \bot \), with the additional restriction\(^3\) that all \( A_{ij} \)'s are distinct and each \( A_{ij} \neq \top \) is equal to some \( B_j \). Modular analytic calculi for them are algorithmically [8] defined (https://www.logic.at/tinc/webexioncalc/) in the hypersequent framework, whose basic objects are disjunctions of sequents which ”work in parallel”, see, e.g., [2, 3].

Neither Hilbert nor hypersequent calculi are good bases for a Curry–Howard correspondence, for different reasons; Hilbert calculi are not analytic, and whereas the hypersequent structure enables analytic proofs it hinders a computational reading of its rules. Our natural deduction calculi are defined by suitably reformulating the corresponding hypersequent calculi, which we describe below. A uniform normalization proof, which works for all of them, will be carried out in Section 5 in the \( \lambda \)-calculus setting.

**Definition 2.1. Hypersequents** are multiset of sequents, written as \( \Gamma_1 \Rightarrow \Pi_1 \downarrow \ldots \downarrow \Gamma_n \Rightarrow \Pi_n \) where, for all \( i = 1, \ldots, n \), \( \Gamma_i \Rightarrow \Pi_i \) is an ordinary sequent, called component.

The symbol ”\( \downarrow \)” is a meta-level disjunction; this is reflected by the presence in hypersequent calculi of the external structural rules (em) and (ec) of weakening and contraction, operating on whole sequents, rather than on formulas.

---

\(^2\)To enforce non-determinism in typing systems for the \( \pi \)-calculus, extensions of linear logic are needed [7].

\(^3\)This restriction enables a simpler notion of analyticity without ruling out interesting computational mechanisms.
Let $\text{Ax}$ be any finite subset of axioms of the form above. A cut-free hypersequent calculus for $\text{IL} + \text{Ax}$ is defined by adding structural rules equivalent to the provability of the axioms in $\text{Ax}$ to the base calculus, i.e. the hypersequent version of the $LJ$ sequent calculus for $\text{IL}$ with (ew) and (ec). These additional rules allow the “exchange of information” between different components. Below we present the (ec) rule and the rule (cl) equivalent to (EM) $A \lor \neg A$:

$$
\frac{G \mid \Gamma \Rightarrow \Pi \mid \Gamma' \Rightarrow \Pi}{G \mid \Gamma, \Gamma' \Rightarrow \Pi} \quad \text{(ec)}
$$

$$
\frac{G \mid \Gamma \Rightarrow \Pi}{G \mid \Gamma \Rightarrow \Pi} \quad \text{(cl)}
$$

In general hypersequent rules equivalent to axiom (1) are [8]

$$
\frac{G \mid \Gamma_1, \Gamma_i \Rightarrow \Pi_i}{G \mid \Gamma_1^i, \ldots, \Gamma_n^i, \Gamma_1 \Rightarrow \ldots \Rightarrow \Gamma_i \Rightarrow \ldots \Rightarrow \Gamma_k \Rightarrow \Pi_k}
$$

for $i, j = 1, \ldots, k$, $p \in \{1, \ldots, n\}$ and possibly $(\Gamma_i, \Pi_i) = \emptyset$. Their reformulation in [9] as higher-level rules à la [26] (see rule 2 in Section 4) is the key for the definition of the natural deduction systems $\text{NAX}$. Some of these rules were considered in [9], [14, 16] but without any normalization proof, which was provided in [1] for a particular instance: the rule for $(A \rightarrow B) \lor (B \rightarrow A)$. Our proof for $\text{NAX}$ is much more general and will treat Gödel logic as a very particular case, also delivering a more expressive set of reductions.

3 Case study: the $\lambda_{\text{EM}}$-calculus

We describe our typed concurrent $\lambda$-calculus $\lambda_{\text{EM}}$ for Classical Logic. Actually, a calculus for IL with axiom $\neg A \lor T \rightarrow A$ can be automatically extracted from the family of calculi $\lambda_{\text{Ax}}$ that we introduce in Section 4. However, the corresponding reduction rules can be significantly simplified and we present here directly the result of the simplification. Since Normalization and Subject Reduction are already proved in Section 5 for the whole family by a very general argument, we shall not bother the reader with the simplified proof: we just remark that the adaption is straightforward.

$\lambda_{\text{EM}}$ extends the standard Curry–Howard correspondence [29] for $\text{NJ}$ with a parallel operator that interprets the natural deduction rule for $A \lor \neg A$. We describe $\lambda_{\text{EM}}$-terms and their computational behavior and show examples of programs.

The table below defines a type assignment for $\lambda_{\text{EM}}$-terms, called proof terms and denoted by $t, u, v,$ . . . , which is isomorphic to $\text{NJ} + \text{EM}$ (see Sec. 1). The typing rules for axioms, implication, conjunction, disjunction and ex-falso-quotidet are those for the simply typed $\lambda$-calculus, while parallelism is introduced by the rule for axiom EM. The contraction rule is useful, although redundant, and is the analogous of the external contraction rule (ec) of hypersequent calculus, see Sec. 2. We fix notation, definitions and terminology that will be used throughout the paper.

Proof terms may contain intuitionistic variables $x_0^A, x_1^A, x_2^A, \ldots$ of type $A$ for every formula $A$; these variables are denoted as $x^A$, $y^A$, $z^A$, . . . . Proof terms also contain channel variables $a_0^A, a_1^A, a_2^A, \ldots$ of type $A$ for every formula $A$; these variables will be denoted as $a^A$, $b^A$, $c^A$, . . . and represent a private communication channel between the parallel processes. Whenever the type is not relevant, it will be dropped and we shall simply write $x, y, z, \ldots, a, b, \ldots$.

Notice that, according to the typing rules, channels variables cannot occur alone and thus are not typed terms, unlike intuitionistic variables. As convention variables $\alpha^A$ will be denoted as $\vec{a}$.

The free and bound variables of a proof term are defined as usual and for the new term $u \|_a v$, all the free occurrences of $a$ in $u$ and $v$ are bound in $u \|_a v$. 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 $\rightarrow$ and $\land$ associate to the right and by $(t_1, t_2, \ldots, t_n)$ we denote the term $(t_1, (t_2, (t_3, \ldots, t_n) \ldots))$ (which is $T$ if $n = 0$) and by $\pi_i$, for $i = 0, \ldots, n$, the sequence $\pi_1, \ldots, \pi_n$ selecting the $(i + 1)\text{th}$ element of the sequence. The expression $A_1 \land \ldots \land A_n$ represents $\land$ if $n = 0$ and thus is the empty conjunction.

Often, if $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and all free variables of a proof term $t : A$ are in $x_1, \ldots, x_n$, we 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 $\text{EM}(t) : A$ whenever $t : A$, and the notation means provability of $A$ in propositional classical logic. If the symbol $||$ does not occur in it, then $t$ is a simply typed $\lambda$-term representing an intuitionistic deduction.

Definition 3.1 (Simple Contexts). A simple context $C[\cdot]$ is a simply typed $\lambda$-term with some fixed variable $\cdot$ occurring exactly once. For any proof term $u$ of the same type of $[\cdot]$, $C[u]$ denotes the term obtained replacing $[\cdot]$ with $u$ in $C[\cdot]$, without renaming of bound variables.

Definition 3.2 (Multiple Substitution). Let $u$ be a proof term, $x = x_0^A, \ldots, x_n^A$ a sequence of variables and $v : A_0 \land \ldots \land A_n$ the substitution $u^{x/x} := u[v_0^{x_0^A} \ldots v_n^{x_n^A}]$ replaces each variable $x_i^A$ of any term $u$ with the $i$th projection of $v$.

We define below the notion of stack, corresponding to Krivine’s stack [20] 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 $\lambda$-calculus perspective, a series of either operations or arguments.

Definition 3.3 (Stack). A stack is a, possibly empty, sequence $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$ such that for every $1 \leq i \leq n$, exactly one of the following holds: either $\sigma_i = \tau$, with $t$ proof term or $\sigma_i = \pi_i$ with $j \in \{0, 1\}$, or $\sigma_i = [x_1 : u_1, x_2 : u_2]$ or $\text{eq}_{\text{b}}$ for some atom $P$. We will denote the empty sequence with $\varepsilon$ and with $\varepsilon, \xi, \xi' \ldots$ the stacks of length 1. If $t$ is a proof term, $t \sigma$ denotes the term $(((t \sigma_1) \sigma_2) \ldots \sigma_n)$.

Definition 3.4 (Case-free). A stack $\sigma$ is case-free if it does not contain any sub-stack of length 1 of the form $[z_1, w_1, z_2, w_2]$. 

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
$x^A : A$ & for $x$ intuitionistic variable & $t_1 : A$ & $t_2 : A$ & contraction \\
\hline
$u : A$ & $t : B$ & $u : A \land B$ & $u : B$ & $u : A \land B$
\hline
$\vdash x^A : A$ & $\vdash x^A : A$ & $\vdash x^A : A$ & $\vdash x^A : A$ & $\vdash x^A : A$
\hline
\end{tabular}
\caption{Table 1. Type assignments for $\lambda_{\text{EM}}$ terms.}
\end{table}
will produce a sequence of message exchanges, ending only when

Definition 3.5 (Parallel Form [1]). A term \( t \) is a parallel form
evertheless define the notion of value, which in our framework will represent a message which is

Definition 3.6 (Value). A value is a term of the form \( \langle t_1, \ldots, t_n \rangle \), where for some \( 1 \leq i \leq n \), \( t_i = \lambda x.s \), \( t_i = t_{i}(s) \), \( t_i = t \text{ if } \varphi \). \( t_i \equiv t \) \if \varphi \). \( t_i \equiv t \) \if \varphi \.

Communication between two processes will take place only when the corresponding channel is active.

Definition 3.7 (Active Channels and Active Sessions). We assume that the set of channel variables is partitioned into two disjoint classes: active channels and inactive channels. A term \( u \parallel v \) is called an active session, whenever \( a \) is active.

A channel can be activated whenever it is applied to at least one value, thus signaling the need for communication.

Definition 3.8 (Activable Channel). We say that \( a \) is an activable channel in \( u \) is there an occurrence \( \mathsf{aw} \) in \( u \) for some value \( v \).

The reduction rules of \( \lambda_{em} \) are in Figure 1. They consist of the familiar reductions for simply typed \( \lambda \)-calculus, instances of \( \lor \) permutations adapted to the \( \mid \) operator, together with new communication reductions.

Activation reductions \( \[ u \parallel v \mapsto u[b/a] \mid \parallel v[b/a] \) can be fired whenever some occurrence of \( a \) in \( u \) or \( v \) is applied to a value: the channel \( a \) is just renamed to an active channel \( b \). This operation has the effect of making the reduce an active session, that will produce a sequence of message exchanges, ending only when all channel occurrences of \( b \) will have transmitted their messages, regardless they are values or not.

Basic Cross reductions \( C[\| u \mid \| D \mapsto D[u/a] \) can be fired whenever the channel \( a \) is active and \( u \) is a closed \( \lambda \)-term. In this case \( u : A \) represents executable code or data and can replace directly all occurrences of the endpoint channel \( a : A \).

Cross Reductions

Surprisingly, logical types enable us to send open \( \lambda \)-terms as messages and to fill their free variables later, when they will be instantiated. Subject reduction will nevertheless be guaranteed. To see how it is possible, suppose that a channel \( a \) is used to send an arbitrary sub-process \( u \) from a process \( s \) to a process \( t \):

Since \( u \) is not necessarily closed, it might depend on its environment \( s \) for some of its inputs - \( y \) in the example is bound by a \( \lambda \) outside \( u \) - but this is solved during the full cross reduction by a fresh channel \( b \) which redirects \( y \) to the new location of \( u \):

The old channel \( a \) is kept for future messages \( s \) might send to \( t \). Technically, the cross reduction has this shape:

\[
C[\| u \mid \| D \mapsto (C[b](y)) \mid \| D \| \| D[u/b/y/a]
\]

We see that, on one hand, the open term \( u \) is replaced in \( C \) by the new channel \( b \) applied to the sequence \( y \) of the free variables of \( u \); on the other hand, \( u \) is sent to \( D \) as \( u/b/y \), so its free variables are removed and replaced by the endpoint channel \( b \) that will receive their future instantiation. In Example 3.2 we exemplify how to use cross reduction for program optimization.
Communication Permutations  The only permutations for \( \parallel \) that are not \( \lor \)-permutation-like are \((u \parallel \langle u \parallel v \rangle)\parallel w \mapsto (u \parallel w)\parallel (v \parallel \langle u \parallel v \rangle)\parallel w\) and \((w \parallel \langle u \parallel v \rangle)\parallel v \mapsto (w \parallel \langle u \parallel v \rangle)\parallel (w \parallel \langle u \parallel v \rangle)\parallel v\). These kind of permutations are between parallel operators themselves and address the scope extrusion issue of private channels [22]. A parallel operator is allowed to commute with another one only when strictly necessary, that is, if there is not already an active session inside \(u, v, w\) that can be first reduced.

The idea behind the normalization (see Section 5 for the proof) is to apply to any term \( t \) the following recursive procedure:

1. Parallel normal form production. We transform \( t \) into a term \( u \) in parallel normal form, using permutations (Prop. 6.1).
2. Intuitionistic Phase. As long as \( u \) contains intuitionistic redexes, we apply intuitionistic reductions.
3. Activation Phase. As long as \( u \) contains activation redexes, we apply activation reductions.
4. Communication Phase. As long as \( u \) contains active sessions, we select the uppermost, permute it upward if necessary or apply directly a cross reduction. Go to step 2.

Since we are dealing with a Curry-Howard isomorphism, every reduction rule of \( \lambda_{EM} \) corresponds to a reduction for the natural deduction calculus \( NJ + (EM) \).

Basic reductions correspond to the logical reductions:

\[
\begin{array}{c|ccc}
\delta & \Pi & [A] & \delta \\
\hline
\lambda & \Gamma & \Lambda & [\Lambda, \Gamma]^* \\
\lambda y. & \Gamma & \lambda y. & [\Lambda, \Gamma]^* \\
\end{array}
\]

where no assumption in \( \delta \) is discharged below \( \bot \) and above \( F \). When this is the case, intuitively, the displayed instance of \((EM)\) is hiding some redex that should be reduced. The reduction precisely expose this potential redex and we are thus able to reduce it.

Cross reductions correspond to the logical reductions:

\[
\begin{array}{c|ccccc}
\delta & \Pi & [A] & \delta \\
\hline
\bot & \Gamma & \Gamma & \bot & [A] & \delta \\
\bot & \Gamma & \bot & \Gamma & \bot & \delta \\
\bot & \bot & \bot & \bot & \bot & \bot \\
\end{array}
\]

Here we use \( \delta \) to prove all assumptions \( A \), as before, but now we also need to discharge the assumptions \( \Gamma \) open in \( \delta \) and discharged below \( \bot \) and above \( F \) in the rightmost branch. This is done by a new (EM) rule application \( * \) to the conjunction \( \land \) of such assumptions. Accordingly, we use the inference \( \land_{\Gamma} \) to replace occurrence of \( \bot_{\Gamma} \) in the leftmost branch. In order to justify the other occurrences of the latter rule, we need the (EM) application joining the leftmost branch and the central branch, which is just a duplicate.

3.2 Computing with \( \lambda_{EM} \)

We present few examples of the computational capabilities of \( \lambda_{EM} \).

Example 3.1 (\( \lambda_{EM} \) vs the simply typed \( \lambda \)-calculus). As is well known there is no \( \lambda \)-term \( O : \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool} \) such that \( \text{OFF} \rightarrow \text{F} \), \( \text{OUT} \rightarrow \text{T} \). We can instead be defined in Boudol’s calculus [5] and in \( \lambda_G \) [1]. Assuming to add the boolean type in our calculus, the \( \lambda_{em} \) term for such parallel OR is

\[ O := \lambda x. \text{Bool}. \lambda y. \text{Bool}. (P_1 \parallel a) (P_2 \parallel b) \]

where \( \langle a : \text{Bool} \rightarrow \bot, a : \text{Bool} \rangle \) and the construct “if \( u \) then \( s \) else \( t \)” is as usual

\[ P_1 = \text{if x then T else F} \]

Example 3.2 (Cross reductions for program efficiency). Consider the parallel processes: \( M \parallel P \parallel (Q \parallel [a]) \). The process \( P \) contains a channel \( a \) to send the message \( (s, y) \) to \( Q \). But the variable \( y \) stands for a missing part of the message which needs to be replaced by a term that \( M \) has to compute and send to \( P \). Hence the whole interaction needs to wait for \( M \). The cross reduction handles precisely this kind of missing arguments. It enables \( P \) to send immediately the message through the channel \( a \) and establishes a new communication channel on the fly which redirects the missing term, when ready, to the new location of the message inside \( Q \). As a concrete example assume that 

\[ \langle a : \text{Bool}, T : \text{Bool} \rangle \]

where \( t \) and \( s \) are closed terms, and the contexts \( M, P \) and \( Q \) do not contain other instances of the channels \( d \) and \( a \). Without a special mechanism for sending open terms, \( P \) must wait for \( M \) to normalize. Afterwards \( M \) passes \( \lambda x. T \) through \( d \) to \( P \) by the following computation:

\[ M \parallel (P \parallel [a]) \longrightarrow^* (M \parallel P) \parallel Q \longrightarrow^* (M \parallel \langle \lambda x. T \rangle) \parallel d \parallel (P \parallel [a]) Q \]

But it is clear that \( Q \) does not need \( t \) at all. Even though \( Q \) waited so long for the pair \((s,t)\), it only keeps the term \( s \).

Our normalization algorithm allows instead \( P \) to directly send \( \lambda y. \langle s, y \rangle \) to \( Q \) by executing a full cross reduction:

\[ M \parallel \langle \lambda y. \langle s, y \rangle \rangle \parallel a \parallel Q \]

where the communication \( b \) handles the redirection of the data \( y \) in case it is available later. But in our case \( Q \) already contains all it needs to terminate its computation, indeed

\[ \rightarrow M \parallel \langle \lambda y. \langle s, b \rangle \rangle \parallel a \parallel Q \parallel [s] \longrightarrow^* Q \parallel [s] \]

since \( Q \parallel [s] \) does not contain communications anymore. Notice that the time-consuming normalization of the term \( M \) does not need to be finished at this point.

3.3 Close relatives of the calculus

As often happens with computational interpretations of logics – compare, for example [13] and [24] – logical equivalence of formulae here does not play a decisive rôle. When extracting communication mechanisms from disjunctive tautologies, differences in the proof-theoretical representation of the axiom can lead to essential differences in the computational behavior of the resulting communication mechanisms. A particular striking case is the class
of formulae with the following shape: \( \neg A \lor A \lor \ldots \lor A \). Indeed, even though all these formulae are equivalent in a very strong sense to (EM), it is very interesting to consider their computational counterparts as distinct objects. Its typing rule is:

\[
\begin{align*}
[\sigma] \cdot A \quad [a : A] \\
\vdots \\
\vdots \\
u \vdash F \\
v_1 \vdash F \\
\vdots \\
v_n \vdash F \\
d(u \vdash v_1 \ldots v_m) \\
\end{align*}
\]

The associated basic cross reduction leads to a broadcasting system:

\[
ad(C[\sigma] u) \parallel D_1 \ldots \parallel D_n \implies D_1[u/a] \parallel \ldots \parallel D_n[u/a]
\]

where \( \sigma : \neg A, a : A, C[\sigma] u \) are simply typed \( \lambda \)-terms, the sequence of free variables of \( u \) is empty, \( a \) does not occur in \( u \), and \( \parallel \) is the operator typed by the contraction rule. The reduction implements a communication schema in which the process corresponding to \( \neg A \) broadcasts the same message to the processes corresponding to the different instances of \( A \).

### 4 General case: the \( \lambda_{\text{Ax}} \) calculi

We introduce the \( \lambda_{\text{Ax}} \) calculi that solve the equation

\[
\frac{\text{simply typed } \lambda \text{-calculus}}{\mathcal{NJ}} = \frac{\text{NAx}}{}
\]

To simplify notation (see remark 5.1), we shall only consider

\[
\lambda_{\text{Ax}} = (A_1 \rightarrow B_1) \lor \ldots \lor (A_m \rightarrow B_m)
\]

where no \( A_i \) is repeated and for every \( A_i \neq \top, A_i = B_j \) for some \( j \).

The set of type assignment rules for \( \lambda_{\text{Ax}} \) terms are obtained replacing the last three rules in Table 1 by

\[
\begin{align*}
u : A & \vdash a^{A_1 \rightarrow B_1} : A_1 \rightarrow B_1 \\
u : B & \vdash a^{A_2 \rightarrow B_2} : A_2 \rightarrow B_2 \\
u : [\ldots] & \vdash a^{A_m \rightarrow B_m} : A_m \rightarrow B_m \\
\end{align*}
\]

all occurrences of \( a \) in \( u_1 \) for \( 1 \leq i \leq m \) are of the form \( a^{A_i \rightarrow B_i} \).

#### 4.1 Reduction rules of \( \lambda_{\text{Ax}} \)

Fig. 2 below shows the reductions for \( \lambda_{\text{Ax}} \)-terms. The permutations rules are simple generalisations of those of \( \lambda_{\text{EM}} \) which only adapt the latter to \( n \)-ary parallelism operators. On the other hand, \( \lambda_{\text{Ax}} \) cross reductions are more complicated than those of \( \lambda_{\text{EM}} \). The formula \( A \) in \( A \lor \neg A \) corresponds indeed to a process that can only receive communications from other processes. The \( \lambda_{\text{Ax}} \) cross reductions cannot rely in all cases on these features and hence require a more general formulation.

**Basic cross reductions** They implement a simple communication of closed programs \( t \)

\[
d(C_1 \parallel \ldots \parallel C_l[A^{E_i \rightarrow G_i} t] \parallel \ldots \parallel C_j[A^{E_j \rightarrow G_j} u] \parallel \ldots \parallel C_m) \iff d(C_1 \parallel \ldots \parallel C_l[A^{E_i \rightarrow G_i} t] \parallel \ldots \parallel C_j[t] \parallel \ldots \parallel C_m)
\]

While the sender \( C_i[a_1 t] \) is unchanged, \( C_i[a_1 u] \) receives the message and becomes \( C_i[t] \). Here, unlike in \( \lambda_{\text{EM}} \) cross reductions, only one receiving channel can be used for each communication. Indeed, if some occurrences of \( a_1 \) are nested, a communication using all of them would break subject reduction.

**Cross Reductions** Basic cross reductions allow non deterministic closed message passing. As in classical logic, cross reductions implement communication with an additional mechanism for handling migrations of open processes. But here each cross reduction application combines at once all possible message exchanges implemented by basic cross reductions, since one cannot know in advance which process will become closed and actually be sent.

For a proof-theoretic view, consider the application of \( \text{Ax} \) (below left), in which all \( \Gamma_i \) for \( 1 \leq i \leq m \) are discharged between \( B_i \) and \( F \). It reduces by full cross reduction to the derivation below right (we explicitly mark with the same label the rule applications belonging to the same higher-level rule):

\[
\begin{align*}
\Gamma_1 & \vdash \Delta_1 \\
\vdots \\
\Delta_m & \vdash \Delta_m \\
\vdots \\
\delta_1 \ldots \delta_m & \vdash F \\
\end{align*}
\]

such that for \( 1 \leq i \leq m \) the derivation \( \delta_i \) is

\[
\begin{align*}
\Gamma_i & \vdash \Delta_i \\
& \vdash u_i \\
\end{align*}
\]

in which \( \delta_j \) is the derivation of the premise \( A_j = B_i \) associated by \( \lambda_{\text{Ax}} \) to \( B_i \) and a double inference line denotes a derivation of its conclusion using the named rule possibly many times.

**Theorem 4.1** (Subject Reduction). If \( t : A \rightarrow t u \), then \( u : A \) and all the free variables of \( u \) appear among those of \( t \).

*Proof.* It is enough to prove the theorem for the cross reductions: if \( t : A \rightarrow t u \), then \( u : A \). The proof that the intuitionistic reductions and the permutation rules preserve the type is completely standard. Basic cross reductions require straightforward considerations as well. Suppose then that

\[
d(C_1[A^{A_1 \ldots \rightarrow B_1} t_1] \parallel \ldots \parallel C_m[A^{A_m \rightarrow B_m} t_m]) \iff d(s_1 \parallel \ldots \parallel s_m)
\]

where \( s_i \) for \( 1 \leq i \leq m \) is

\[
d(C_1[A^{A_1 \ldots \rightarrow B_1} t_1] \parallel \ldots \parallel C_i[t_j^{B_i \rightarrow B_j}(y_i)/y_i] \parallel \ldots \parallel C_m[A^{A_m \rightarrow B_m} t_m])
\]

Since \( (y_i) : B_i, b^{B_i \rightarrow B_j} \) are correct terms, and hence \( t_j^{B_i \rightarrow B_j}(y_i)/y_i \), by Definition 3.2, are correct as well. The assumptions are that \( t_j^{B_i \rightarrow B_j}(y_i)/y_i \) has the same type as \( a^{A_i \rightarrow B_j} t_j \); \( y_j \) for \( 1 \leq i \leq m \) is the sequence of the free variables of \( t_j \) which are bound in \( C_i[A^{A_i \rightarrow B_j} t_i] \); \( B_i \) for \( 1 \leq i \leq m \) is the conjunction of the types of the variables in \( y_i \); \( a \) is rightmost in each \( C_i[A^{A_i \rightarrow B_j} t_i] \); and \( b \) is fresh. Hence, by construction all the variables \( y_i \) are bound in each \( C_i[t_j^{B_i \rightarrow B_j}( y_i )/y_i] \). Hence, no new free variable is created.

**Example 4.1** (Gödel Logic). A particular instance of \( \lambda_{\text{Ax}} \) is defined by \( \text{Ax} = (A \rightarrow B) \lor (B \rightarrow A) \). The resulting type assignment rules and reductions are those of \( \lambda_{\text{Ax}} \) [1] but for the activation conditions of these, which are no more based on the types.

For the Subformula Property, we shall need the following definition.

**Definition 4.1** (Prime Formulae and Factors [19]). A formula is said to be prime if it is not a conjunction. Every formula is a conjunction of prime formulae, called prime factors.
where each $t_a$ is a fresh $F_j$-variable.

Parallel Operator Permutations

where the set of channel variables is partitioned into two disjoint

Definition 4.4

Disjunctive Axioms and Concurrent Calculi: a Curry-Howard Approach Technical report, ArXiv, 2018

We use them to implement a cyclic scheduler along the lines of that in [21]. Such scheduler is designed to ensure that a certain group of processes performs the assigned tasks in cyclic order. Each process can perform its present task in parallel with each other, and each of them can stop its present round of computation at any moment, but no process should start its nth round of computation before its predecessor has started the respective nth round of computation.

Consider the processes $C[a_1(r(a_1t'))], D[a_2(s(a_2s'))]$, and $E[a_3(t(a_3t'))]$ where $a_1 : A \to B$, $a_2 : C \to A$ and $a_3 : B \to C$. We implement the scheduling program as the following $\lambda_C$-term

\[
\xi(C[a_1(r(a_1r'))] \times D[a_2(s(a_2s'))] \times E[a_3(t(a_3t'))])
\]

when $r'$ is over with its computation, according to the strategy in Def. 5.2, this term reduces to

\[
\xi(C[a_1(r(a_1r'))] \times D[a_2(s(r'))] \times E[a_3(t(a_3t'))])
\]

then, after the evaluation of $s$, to

\[
\xi(C[a_1(r(a_1r'))] \times D[a_2(s(r'))] \times E[a_3(t(s(a_3t')))])
\]

and after the evaluation of $t$ to

\[
\xi(C[a_1(r(t(s(a_3t'))))] \times D[a_2(s(r'))] \times E[a_3(t(s(r')))])
\]

and so on until all arguments of channels $a_i$ for $i \in \{1, 2, 3\}$ have normalized and have been communicated.

5 The Normalization Theorem

Our goal is to prove the Normalization Theorem for $\lambda_A$: every proof term of $\lambda_A$ reduces in a finite number of steps to a normal form. By Subject Reduction, this implies that the corresponding natural deduction proofs do normalize. We shall define a reduction strategy for terms of $\lambda_A$: a recipe for selecting, in any given term, the subterm to which apply one of our basic reductions.

The idea behind our normalization strategy is quite intuitive. We start from any term and reduce it in parallel normal form, thanks to Proposition 5.2. Then we cyclically interleave three reduction...
phases. First, an \textit{intuitionistic phase}, where we reduce all intuitionistic redexes. Second, an \textit{activation phase}, where we activate all sessions that can be activated. Third, a \textit{communication phase}, where we allow the active sessions to exchange messages as long as they need and we enable the receiving process to extract information from the messages. Technically, we perform all cross reductions combined with the generated structural intuitionistic redexes, which we consider to be projections and case permutations.

Proving termination of this strategy is by no means easy, as we have to rule out two possible infinite loops.

1. Intuitionistic reductions can generate new activable sessions that want to transmit messages, while message exchanges can generate new intuitionistic reductions.
2. During the communication phase, new sessions may be generated after each cross reduction and old sessions may be duplicated after each session permutation. The trouble is that each of these sessions may need to send new messages, for instance forwarding some message received from some other active session. So the count of active sessions might increase forever and the communication phase never terminate.

We break the first loop by focusing on the complexity of the exchanged messages. Since messages are values, we shall define a notion of \textit{value complexity} (Definition 5.3), which will simultaneously ensure that: (i) after firing a non-structural intuitionistic redex, the new active sessions can ask to transmit only new messages of smaller value complexity than the complexity of the fired redex; (ii) after transmitting a message, all the new generated intuitionistic reductions have at most the value complexity of the message. Proposition 5.6 will settle the matter, but in turn requires a series of preparatory lemmas. Namely, we shall study how arbitrary substitutions affect the value complexity of a term in Lemma 5.5 and Lemma 5.6; then we shall determine how case reductions impact value complexity in Lemma 5.8 and Lemma 5.7.

We break the second loop by showing in the crucial Lemma 5.10 that message passing, during the communication phase, cannot produce new active sessions. Intuitively, the new generated channels and the old duplicated ones are “frozen” and only intuitionistic reductions can activate them, thus with values of smaller complexity than that of the fired redex.

For clarity, we define here the recursive normalization algorithm that represents the constructive content of this section’s proofs, which are used to prove the Normalization Theorem. Essentially, our master reduction strategy will use in the activation phase the basic reduction relation \(\parallel\) defined below, whose goal is to permute an uppermost active session \(a(u_1 \parallel \ldots \parallel u_m)\) until all \(u_i\) for \(1 \leq i \leq m\) are simply typed \(\lambda\)-terms and finally apply the cross reductions followed by projections and case permutations.

\textbf{Definition 5.1 (Side Reduction Strategy).} Let \(t\) be a term and \(a(u_1 \parallel \ldots \parallel u_m)\) be an active session occurring in \(t\) such that no active session occurs in \(u\) or \(v\). We write

\[ t \parallel a \quad t' \parallel a \]

whenever \(t'\) has been obtained from \(t\) by applying to \(u \parallel a \parallel v\):

1. a permutation reduction

\[
a(u_1 \parallel \ldots \parallel u_n) \parallel \lambda(w_1 \parallel \ldots \parallel w_n) \quad 7\rightarrow \lambda(w_1 \parallel \ldots \parallel w_n) \parallel a(u_1 \parallel \ldots \parallel u_n)
\]

if \(u_i = \lambda w_i \parallel \ldots \parallel \lambda w_n\) for some \(1 \leq i \leq m\);

2. a cross reduction, if \(u_1, \ldots, u_m\) are intuitionistic terms, immediately followed by the projections and case permutations inside the newly generated simply typed \(\lambda\)-terms;

3. a cross reduction \(\lambda(u_1 \parallel \ldots \parallel u_m) \parallel \lambda(u_{1'} \parallel \ldots \parallel u_{m'}) \quad 7\rightarrow \lambda(u_{1'} \parallel \ldots \parallel u_{m'}) \parallel \lambda(u_1 \parallel \ldots \parallel u_m)\) for \(1 \leq j_1 < \ldots < j_n \leq m\), if \(a\) does not occur in \(u_{j_1}, \ldots, u_{j_n}\).

\textbf{Definition 5.2 (Master Reduction Strategy).} Let \(t\) be any term which is not in normal form. We transform it into a term \(u\) in parallel form, then we execute the following three-step recursive procedure.

1. \textit{Intuitionistic Phase}. As long as \(u\) contains intuitionistic redexes, we apply intuitionistic reductions.
2. \textit{Activation Phase}. As long as \(u\) contains activation redexes, we apply activation reductions.
3. \textit{Communication Phase}. As long as \(u\) contains active sessions, we apply the Side Reduction Strategy (Definition 5.1) to \(u\), then we go to step 1.

We start defining the value complexity of messages. Intuitively, it is a measure of the complexity of the redexes that a message can generate after being transmitted. On one hand, it is defined as usual for proper values, like \(s = \lambda xu, s = \iota u\), as the complexity of their types. On the other hand, pairs \((u, v)\) and case distinctions \((x, u, y, v)\) represent sequences of values, hence we pick recursively the maximum among the value complexities of \(u\) and \(v\). This is a crucial point. If we chose the types as value complexities also for pairs and case distinctions, then our argument would completely break down when new channels are generated during cross reductions: their type can be much bigger than the starting channel and any shade of a decreasing complexity measure would disappear.

\textbf{Definition 5.3 (Value Complexity).} For any simply typed \(\lambda\)-term \(s : T\), the value complexity of \(s\) is defined as the first case that applies among the following:

- if \(s = \lambda xu, s = \iota u\), then the value complexity of \(s\) is the complexity of its type \(T\);
- if \(s = (u, v)\), then the value complexity of \(s\) is the maximum among the value complexities of \(u\) and \(v\);
- if \(s = t(x, u, y, v)\sigma\) where \(\sigma\) is case-free, then the value complexity of \(s\) is the maximum among the value complexities of \(u\sigma\) and \(v\sigma\);
- otherwise, the value complexity of \(s\) is 0.

Recall that values are defined (Def. 3.6) as anything that either can generate an intuitionistic redex when plugged into another term or that can be transformed into something with that capability, like an active channel acting as an endpoint of a transmission.

The value complexity of a term, as expected, is always at most the complexity of its type.

\textbf{Proposition 5.1.} Let \(u : T\) be any simply typed \(\lambda\)-term. Then the value complexity of \(u\) is at most the complexity of \(T\).

\textbf{Proof.} Induction on the shape of \(u\). See Appendix. \(\square\)

The complexity of an intuitionistic redex \(t_f^j\) is defined as the value complexity of \(t\).

\textbf{Definition 5.4 (Complexity of the Intuitionistic Redexes).} Let \(r\) be an intuitionistic redex. The complexity of \(r\) is defined as follows:

- If \(r = (\lambda xu)\), then the complexity of \(r\) is the type of \(\lambda xu\).
By induction on the number of non-active terms of the form $t_1(t)$. 
- if $r = t(x.u)$ then the complexity of $r$ is the type of $t_2(t)$. 
- if $r = t(x,u,v)$ then the complexity of $r$ is the value complexity of $(u,v)$. 
- if $r = t(x,u,v,w)$ then the complexity of $r$ is the value complexity of $t(x,u,v,w)$. 

The value complexity is used to define the complexity of communication redexes. Intuitively, it is the value complexity of the heaviest message ready to be transmitted.

**Definition 5.5** (Complexity of the Communication Redexes). Let $u \parallel_0 v : A$ a term. Assume that $a^{B \rightarrow C}$ occurs in $u$ and $a^{C \rightarrow B}$ in $v$.
- The pair $B, C$ is called the **communication kind of $a$**.
- The **complexity of a channel occurrence** $a(t_1, \ldots, t_n)$ is the value complexity of $(t_1, \ldots, t_n)$ (see Definition 5.3).
- The **complexity of a communication redex** $u \parallel_0 v$ is the maximum among the complexities of the occurrences of $a$ in $u$ and $v$.
- The **complexity of a permutation redex** $\sigma_{(w_1, \ldots, w_n)}(u \parallel_0 v)$ is $0$.

As our normalization strategy suggests, application and injection redexes should be treated differently from the others, because they generate the real computations.

**Definition 5.6.** We distinguish two groups of redexes:
1. Group 1: Application and injection redexes.
2. Group 2: Communication redexes, projection redexes and case permutation redexes.

The first step of the normalization proof consists in showing that any term can be reduced to a parallel form.

**Proposition 5.2** (Parallel Form). Let $t : A$ be any term. Then $\mapsto^* t'$, where $t'$ is a parallel form.

**Proof.** Similar to [1]. See Appendix. □

The following, easy lemma shows that the activation phase of our reduction strategy is finite.

**Lemma 5.3** (Activate!). Let $t$ be any term in parallel form that does not contain intuitionistic redexes and whose communication redexes have complexity at most $\tau$. Then there exists a finite sequence of activation reductions that results in a term $t'$ that contains no redexes, except cross reduction redexes of complexity at most $\tau$.

**Proof.** By induction on the number of non-active terms of the form $\sigma_{(u_1, \ldots, u_m)}(t)$. See Appendix. □

We shall need a simple property of the value complexity notion.

**Lemma 5.4** (Why Not 0). Let $u$ be any simply typed $\lambda$-term and $\sigma$ be a non-empty case-free stack. Then the value complexity of $u \sigma$ is 0.

**Proof.** Induction on the size of $u$. See Appendix. □

In order to formally study redex contraction, we consider simple substitutions that just replace some occurrences of a term with another, allowing capture of variables. In practice, it will always be clear from the context which occurrences are replaced.

**Definition 5.7** (Simple Replacement). By $s(t/u)$ we denote a term obtained from $s$ be replacing some occurrences of a term $u$ with a term $t$ of the same type of $u$, possibly causing capture of variables.

We now show an important property of value complexity: the value complexity of $w/s$ either remains at most as it was before the substitution or becomes exactly the value complexity of $v$.

**Lemma 5.5** (The Change of Value). Let $w, s, v$ be simply typed $\lambda$-terms with value complexity respectively $0, \tau, \tau'$. Then the value complexity of $w/s$ is either at most $\tau$ or equal to $\tau'$. Moreover, if $\tau' \leq \tau$, then the value complexity of $w/s$ is at most $\tau$.

**Proof.** Induction on the size of $w$. See Appendix. □

The following lemma has the aim of studying mostly message passing and contraction of application and projection redexes.

**Lemma 5.6** (Replace!). Let $u$ be a term in parallel form, $v, s$ be any simply typed $\lambda$-terms, $\tau$ be the value complexity of $u$ and $\tau'$ be the maximum among the complexities of the channel occurrences in $v$. Then every redex in $u/vs$ is either (i) already in $v$, (ii) of the form $r/vs$, and has complexity smaller than or equal to the complexity of some redex $r$ of $u$, or (iii) has complexity $\tau$ or is a communication redex of complexity at most $\tau'$.

**Proof.** Induction on the size of $u$. See Appendix. □

Below we study the complexity of redexes generated after contracting an injection redex.

**Lemma 5.7** (Eliminate the Case!). Let $u$ be a term in parallel form. Then for any redex $r$ in $u/w_1/t/x_1/x_i(t_1, w_1, x_2, w_2)$ of complexity $\theta$, either $u/t/x_1/x_2/w_1/w_2$ has complexity greater than $\theta$; or there is a redex in $u$ of complexity $\theta$ which belongs to the same group as $r$ or is a case permutation redex.

**Proof.** Induction on the size of $u$. See Appendix. □

We analyze what happens after permuting a case distinction.

**Lemma 5.8** (In Case!). Let $u$ be a term in parallel form. Then for any redex $r_1$ of Group 1 in $u/w_1/t/x_1/x_i(t_1, x_1, x_2, x_3, y_1, y_2)$ of complexity $\theta_1$, either $u/t/x_1/x_2/w_1/w_2$ has complexity greater or equal to $\theta_1$; for any redex $r_2$ of Group 2 in $u/t/x_1/x_2/x_3/x_4/x_5$ of complexity $\theta_2$, there is a redex of Group 2 in $u$ with greater or equal complexity than $\theta_2$.

**Proof.** Induction on the size of $u$. See Appendix. □

The following result is meant to break the possible loop between the intuitionistic phase and communication phase of our normalization strategy. Intuitively, when Group 1 redexes generate new redexes, these latter have smaller complexity than the former; when Group 2 redexes generate new redexes, these latter does not have worse complexity than the former.

**Proposition 5.9** (Decrease!). Let $t$ be a term in parallel form, $r$ be one of its redexes of complexity $\tau$, and $t'$ be the term that we obtain from $t$ by contracting $r$.

1. If $r$ is a redex of the Group 1, then the complexity of $t'$ is at most the complexity of $r$ of the same group and occurring in $t$; or is at most the complexity of a case permutation redex occurring in $t$; or is smaller than $\tau$.
2. If $r$ is a redex of the Group 2 and not an activation redex, then every redex in $t'$ either has the complexity of a redex of the same group occurring in $t$ or has complexity at most $\tau$.

**Proof.** See Appendix. □
The following result is meant to break the possible loop during the communication phase: no new activation is generated after a cross reduction, when there is none to start with.

**Lemma 5.10 (Freeze!).** Suppose that \( s \) is a term in parallel form that does not contain projection nor case permutation nor activation redexes. Let \( d(q_1 \parallel \ldots \parallel q_m) \) be some redex of \( s \) of complexity \( \tau \). If \( s' \) is obtained from \( s \) by performing first a cross reduction on \( d(q_1 \parallel \ldots \parallel q_m) \) and then contracting all projection and case permutation redexes, then \( s' \) contains no activation redexes.

**Proof.** We reason on the simple replacements produced by cross reductions, case permutations and projections. See Appendix.

**Definition 5.8.** The height \( h(t) \) of a term \( t \) in parallel form is
- \( h(u) = 0 \) if \( u \) is simply typed \( \lambda \)-term
- \( h(u \parallel u v) = 1 + \max(h(u), h(v)) \)

The communication phase of our reduction strategy is finite.

**Lemma 5.11 (Communicate!).** Let \( t \) be any term in parallel form that does not contain projection, case permutation, or activation redexes. Assume moreover that all redexes in \( t \) have complexity at most \( \tau \). Then \( t \) reduces to a term containing no redexes, except Group 1 redexes of complexity at most \( \tau \).

**Proof.** We reason by lexicographic induction on a triple defined using the number of non-uppermost active sessions in \( t \), and the height and number of channels of the uppermost active sessions in \( t \). See Appendix.

We now combine together all the main results achieved so far.

**Proposition 5.12 (Normalize!).** Let \( t : A \) be any term in parallel form. Then \( t \mapsto^* t' \), where \( t' \) is a parallel normal form.

**Proof.** Let \( \tau \) be the maximum among the complexity of the redexes in \( t \). We prove the statement by induction on \( \tau \). Starting from \( t \), we reduce all intuitionistic redexes and obtain a term \( t_1 \) that, by Proposition 5.9, does not contain redexes of complexity greater than \( \tau \). By Lemma 5.3, \( t_1 \mapsto^* t_2 \) where \( t_2 \) does not contain any redex, except cross reduction redexes of complexity at most \( \tau \). By Lemma 5.11, \( t_2 \mapsto^* t_3 \) where \( t_3 \) contains only Group 1 redexes of complexity at most \( \tau \). Suppose \( t_3 \mapsto^* t_4 \) by reducing all Group 1 redexes, starting from \( t_3 \). By Proposition 5.9, every Group 1 redex generated in the process has complexity at most \( \tau \), thus every Group 2 redex which is generated has complexity smaller than \( \tau \), thus \( t_4 \) can only contain redexes with complexity smaller than \( \tau \). By induction hypothesis \( t_4 \mapsto^* t' \), with \( t' \) in parallel normal form.

The normalization for \( \lambda_{AX} \) now easily follows.

**Theorem 5.13.** Suppose that \( t : A \) is a \( \lambda_{AX} \) proof-term. Then \( t \mapsto^* t' : A \), where \( t' \) is a normal parallel form.

**Remark 5.1.** This normalization proof covers all systems corresponding to axioms in the class (1) but the notation of typing rules and reductions becomes much more involved.

6 The Subformula Property

We show that normal \( \lambda_{AX} \)-terms satisfy the Subformula Property: a normal proof does not contain concepts that do not already appear in the premisses or in the conclusion. This, in turn, implies that our Curry–Howard correspondence for \( \lambda_{AX} \) is meaningful from the logical perspective and produces analytic proofs.

**Proposition 6.1 (Parallel Normal Form Property).** If \( t \in NF \) is a \( \lambda_{AX} \)-term, then it is in parallel form.

**Proof.** Easy induction on the shape of \( t \).

**Theorem 6.2 (Subformula Property).** Suppose

\[
x_1, \ldots, x_n, D_1, \ldots, D_m \vdash t : A \quad \text{and} \quad t \in NF.
\]

Then:

1. For each channel variable \( a \parallel B \rightarrow C \) occurring bound in \( t \), the prime factors of \( B, C \) are subformulas of \( A_1, \ldots, A_n, A \) or proper subformulas of \( D_1, \ldots, D_m \).
2. The type of any subterm of \( t \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) and of proper subformulas of \( D_1, \ldots, D_m \).

**Proof.** Induction on the shape of \( t \). See Appendix.

References

[1] F. Aschieri, A. Ciabattoni and F.A. Genco. Gödel logic: From Natural Deduction to Parallel Computation. LICS, 1–12, 2017.
[2] A. Avron. Hypersequents, Logical Consequence and Intermediate Logics for Concurrency. Annals of Mathematics and Artificial Intelligence, 4: 225–248, 1991.
[3] A. Avron. The Methods of Hypersequents in the Proof Theory of Propositional Non-classical Logic. Logic: From Foundations to Applications, 1–32, 1996.
[4] A. Beckmann and N. Preising. Hyper Natural Deduction. LICS, 547–558, 2015.
[5] G. Boudol. Towards a Lambda-calculus for Concurrent and Communicating Systems. TAPSOFT, 1: 149-161, 1989.
[6] L. Caires and F. Penning. Session Types as Intuitionistic Linear Propositions. CONCUR, 222-236, 2010.
[7] L. Caires and J. Pérez. Linearity, Control Effects, and Behavioral Types. ESOP, 229-259, 2017.
[8] A. Ciabattoni, N. Galatos and K. Terui. From Axioms to Analytic Rules in Non-classical Logics. LICS, 229–240, 2008.
[9] A. Ciabattoni and F.A. Genco. Hypersequents and Systems of Rules: Embeddings and Applications. Accepted for publication in ACM TOCL.
[10] V. Danos, J.-L. Krivine. Disjunctive Tautologies as Synchronisation Schemes. CSL, 292–301, 2000.
[11] A. Fuggetta, G.P. Picco and G. Vigna. Understanding Code Mobility. IEEE Transactions on Software Engineering, 24: 342–361, 1998.
[12] J.-Y. Girard and Y. Lafont and P. Taylor. Proofs and Types. Cambridge University Press, 1989.
[13] T. Griffin. A Formule-as-Type Notion of Control. POPL, 1990.
[14] P. de Groote. A Simple Calculus of Exception Handling. Proceedings of TLCA, 902: 201–215, 1995.
[15] M. Felleisen and R. Hieb. The Revised Report on the Syntactic Theories of Sequential Control and State. Theor. Comput. Sci., 103(2): 235–271, 1992.
[16] E.G.K. Lopez-Escobar. Implicational Logics in Natural Deduction Systems. J. Symbolic Logic, 47(1): 184–186, 1982.
[17] Y. Hirai. A Lambda Calculus for Gödel–Dummett Logic Capturing Waitfreedom. FLOPS 2012, 151–165, 2012.
[18] W.A. Howard. The Formule-as-Type Notion of Construction. In To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, Academic Press, pp. 479–491, 1980.
[19] J.-L. Krivine. Lambda-calcul: types et modèles. In Studies in Logic and Foundations of Mathematics, 1–176, 1990.
[20] J.-L. Krivine. Classical Realizability. Interactive models of computation and program behavior, Panoramas et synthèses, 197–229, 2009.
[21] R. Milner. Communication and Concurrency. Prentice Hall, 1989.
[22] R. Milner. Functions as Processes. Mathematical Structures in Computer Science, 2(2): 119–141, 1992.
[23] J. Niehren, J. Schwinghammer and G. Smolka. A Concurrent Lambda Calculus with Futures. Theoretical Computer Science, 364(3): 338–356, 2006.
[24] M. Parigot. Proofs of Strong Normalization for Second-Order Classical Natural Deduction. Journal of Symbolic Logic, 62(4): 1461–1479, 1997.
[25] D. Prawitz. Ideas and Results in Proof Theory. In Proceedings of the Second Scandinavian Logic Symposium, 1971.
A The Normalization Theorem

Our goal is to prove the Normalization Theorem for $\lambda_{Ax}$: every proof term of $\lambda_{Ax}$ reduces in a finite number of steps to a normal form. By Subject Reduction, this implies that the corresponding natural deduction proofs do normalize. We shall define a reduction strategy for terms of $\lambda_{Ax}$: a recipe for selecting, in any given term, the subterm to which apply one of our basic reductions. As opposed to [1], our permutations between communications do not enable silly loops and thus do not undermine the possibility that our set of reduction is strongly normalizing. We leave anyway as a difficult, open problem to determine whether this is really the case.

The idea behind our normalization strategy is quite intuitive. We start from any term and reduce it in parallel normal form, thanks to Proposition 5.2. Then we cyclically interleave three reduction phases. First, an intuitionistic phase, where we reduce all intuitionistic redexes. Second, an activation phase, where we activate all sessions that can be activated. Third, a communication phase, where we allow the active sessions to exchange messages as long as they need and we enable the receiving process to extract information from the messages. Technically, we perform all cross reductions combined with the generated structural intuitionistic redexes, which we consider to be projections and case permutations.

Proving termination of this strategy is by no means easy, as we have to rule out two possible infinite loops.

1. Intuitionistic reductions can generate new activable sessions that want to transmit messages, while message exchanges can generate new intuitionistic reductions.

2. During the communication phase, new sessions may be generated after each cross reduction and old sessions may be duplicated after each session permutation. The trouble is that each of these sessions may need to send new messages, for instance forwarding some message received from some other active session. Thus the count of active sessions might increase forever and the communication phase never terminate.

We break the first loop by focusing on the complexity of the exchanged messages. Since messages are values, we shall define a notion of value complexity (Definition 5.3), which will simultaneously ensure that: (i) after firing a non-structural intuitionistic redex, the new active sessions can ask to transmit only new messages of smaller value complexity than the complexity of the fired redex; (ii) after transmitting a message, all the new generated intuitionistic reductions have at most the value complexity of the message. Proposition 5.6) will settle the matter, but in turn requires a series of preparatory lemmas. Namely, we shall study how arbitrary substitutions affect the value complexity of a term in Lemma 5.5 and Lemma 5.6; then we shall determine how case reductions impact value complexity in Lemma 5.8 and Lemma 5.7.

We break the second loop by showing in the crucial Lemma 5.10 that message passing, during the communication phase, cannot produce new active sessions. Intuitively, the new generated channels or the old duplicated ones are “frozen” and only intuitionistic reductions can activate them, thus with values of smaller complexity than that of the fired redex.

For clarity, we define here the recursive normalization algorithm that represents the constructive content of this section’s proofs, which are used to prove the Normalization Theorem. Essentially, our master reduction strategy will use in the activation phase the...
basic reduction relation \( > \) defined below, whose goal is to permute an uppermost active session \( a(u_1 \ldots u_m) \) until all \( u_i \) for \( 1 \leq i \leq m \) are simply typed \( \lambda \)-terms and finally apply the cross reductions followed by projections and case permutations.

**Definition A.1** (Side Reduction Strategy). Let \( t \) be a term and \( a(u_1 \ldots u_m) \) be an active session occurring in \( t \) such that no active session occurs in \( u \) or \( v \). We write

\[ t > t' \]

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

1. a permutation reduction

\[ a(u_1 \ldots u_m) \rightarrow b(u_1 \ldots u_m) \]

if \( u_i = b(w_i) \) for some \( 1 \leq i \leq m \);

2. a cross reduction, if \( u_1, \ldots, u_m \) are intuitionistic terms, immediately followed by the projections and case permuta-

3. a cross reduction \( a(u_1 \ldots u_m) \rightarrow b(u_1 \ldots u_m) \)

if \( 1 \leq j_1 < \cdots < j_n \leq m \), if \( u_{j_k} \) does not occur in \( u_{j_1}, \ldots, u_{j_n} \)

**Definition A.2** (Master Reduction Strategy). Let \( t \) be any term which is not in normal form. We transform it into a term \( u \) in parallel form, then we execute the following three-step recursive procedure.

1. **Intuitionistic Phase.** As long as \( u \) contains intuitionistic redexes, we apply intuitionistic reductions.
2. **Activation Phase.** As long as \( u \) contains activation redexes, we apply activation reductions.
3. **Communication Phase.** As long as \( u \) contains active sessions, we apply the Side Reduction Strategy (Definition 5.1) to \( u \), then we go to step 1.

We start be defining the value complexity of messages. Intuitive, it is a measure of how much complex redexes a message can generate, after being transmitted. On one hand, it is defined as usual for true values, like \( s = \lambda x u, s = \iota_1(u) \), as the complexity of their types. On the other hand, pairs \( (u, v) \) and case distinctions \( t[x,u,y,v] \) represents sequences of values, therefore we pick recur-

**Definition A.3** (Value Complexity). For any simply typed \( \lambda \)-term \( s : T \), the value complexity of \( s \) is defined as the first case that applies among the following:

- If \( s = \lambda x u, s = \iota_1(u) \), then the value complexity of \( s \) is the complexity of its type \( T \);
- If \( s = (u, v) \), then the value complexity of \( s \) is the maximum

among the value complexities of \( u \) and \( v \);
- If \( s = t[x,u,y,v] \sigma \), where \( \sigma \) is case-free, then the value complexity of \( s \) is the maximum among the value complexities of \( u \sigma \) and \( v \sigma \);
- otherwise, the value complexity of \( s \) is 0.

Values are defined as anything that either can generate an intuitionistic redex when plugged into another term or that can be transformed into something with that capability, like an active channel acting as an endpoint of a transmission.

**Definition A.4** (Value). A **value** is a term of the form \( \iota_1 \ldots \iota_n \), for some \( 1 \leq i \leq n \), \( t = \lambda x u, s = \iota_1(s), t = t[x,u,y,v], t \) or \( t = a \sigma \) for an active channel \( a \).

The value complexity of a term, as expected, is always at most the complexity of its type.

**Proposition A.1.** Let \( u : T \) be any simply typed \( \lambda \)-term. Then the value complexity of \( u \) is at most the complexity of \( T \).

**Proof.** By induction on \( u \). There are several cases, according to the shape of \( u \):

- If \( u \) is of the form \( \lambda x w, \iota_1(w) \), then the value complexity of \( u \) is indeed the complexity of \( T \).
- If \( u \) is of the form \( \langle v_1, v_2 \rangle \) then, by induction hypothesis, the value complexities of \( v_1 \) and \( v_2 \) are at most the complexity of their respective types \( T_1 \) and \( T_2 \), and hence at most the complexity of \( T = T_1 \land T_2 \), so we are done.

- If \( u \) is of the form \( \tau_0 [v_1, v_2, v_3] \) then, by induction hypothesis, the value complexities of \( v_1 \) and \( v_2 \) at most the complexity of \( T \), so we are done.
- In all other cases, the value complexity of \( u \) is 0, which is trivially the thesis.

The complexity of an intuitionistic redex \( t \xi \) is defined as the value complex-

**Definition A.5** (Complexity of the Intuitionistic Redexes). Let \( r \) be an intuitionistic redex. The complexity of \( r \) is defined as follows:

- If \( r = \lambda x u \), then the complexity of \( r \) is the type of \( \lambda x u \).

- If \( r = \iota_1(t)[x,u,y,v] \), then the complexity of \( r \) is the type of \( \iota_1(t) \).

- If \( r = (u, v) \pi_1 \) then the complexity of \( r \) is the value complex-

- If \( r = t[x,u,y,v] \xi \), then the complexity of \( r \) is the value complex-

The value complexity is used to define the complexity of communication redexes. Intuitively, it is the value complexity of the heavies message ready to be transmitted.

**Definition A.6** (Complexity of the Communication Redexes). Let \( u \parallel_a v : A \) a term. Assume that \( a^{B \rightarrow C} \) occurs in \( u \) and thus \( a^{C \rightarrow B} \) in \( v \).

- The pair \( B, C \) is called the communication kind of \( a \).

The **complexity of a channel occurrence** \( a \langle t_1, \ldots, t_n \rangle \) is the value complex-

The **complexity of a communication redex** \( u \parallel_a v \) is the maximum among the complexities of the occurrences of \( a \) in \( u \) and \( v \).

The **complexity of a permutation redex**

\[ a(u_1 \ldots u_m) \rightarrow b(u_1 \ldots u_m) \]

is 0.

As our normalization strategy suggests, application and injection redexes should be treated differently from the others, because generate the real computations.
Definition A.7. We distinguish two groups of redexes:
1. Group 1: Application and injection redexes.
2. Group 2: Communication redexes, projection redexes and case permutation redexes.

The first step of the normalization proof consists in showing that any term can be reduced to a parallel form.

Proposition A.2 (Parallel Form). Let t : A be any term. Then t \rightarrow^{*} t', where t' is a parallel form.

Proof. By induction on t. As a shortcut, if a term t reduces to a term u' that can be denoted as u'' omission of parentheses, we write t \equiv^{*} u''.

- t is a variable x. Trivial.
- t = \lambda x u. By induction hypothesis, u \equiv^{*} u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1} and each term u_{i}, for 1 \leq i \leq n + 1, is a simply typed \lambda-term. Applying several permutations we obtain
  t \equiv^{*} \lambda x u_{1} \parallel \lambda x u_{2} \parallel \ldots \parallel \lambda x u_{n+1}
  which is the thesis.
- t = u \parallel v. By induction hypothesis,
  u \equiv^{*} u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1}
  v \equiv^{*} v_{1} \parallel v_{2} \parallel \ldots \parallel v_{m+1}
  and each term u_{i} and v_{i}, for 1 \leq i \leq n + m + 1, is a simply typed \lambda-term. Applying several permutations we obtain
  t \equiv^{*} (u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1} \parallel v)
  \equiv^{*} (u_{1} \parallel v_{1} \parallel u_{2} \parallel v_{2} \parallel \ldots \parallel u_{n+1} \parallel v_{m+1})
  \equiv^{*} (u_{1} \parallel v_{1} \parallel u_{2} \parallel v_{2} \parallel \ldots \parallel u_{n+1} \parallel v_{m+1})
  \equiv^{*} (u_{1} \parallel v_{1} \parallel u_{2} \parallel v_{2} \parallel \ldots \parallel u_{n+1} \parallel v_{m+1})
  \equiv^{*} (u_{1} \parallel v_{1} \parallel u_{2} \parallel v_{2} \parallel \ldots \parallel u_{n+1} \parallel v_{m+1})
- t = (u, v). By induction hypothesis,
  u \equiv^{*} u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1}
  v \equiv^{*} v_{1} \parallel v_{2} \parallel \ldots \parallel v_{m+1}
  and each term u_{i} and v_{i}, for 1 \leq i \leq n + m + 1, is a simply typed \lambda-term. Applying several permutations we obtain
  t \equiv^{*} (u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1}, v)
  \equiv^{*} (u_{1}, v) \parallel (u_{2}, v) \parallel \ldots \parallel (u_{n+1}, v)
  \equiv^{*} (u_{1}, v_{1}) \parallel (u_{2}, v_{2}) \parallel \ldots \parallel (u_{1}, v_{m+1}) \parallel \ldots \parallel (u_{n+1}, v_{m+1})
  \equiv^{*} (u_{1}, v_{1}) \parallel (u_{2}, v_{2}) \parallel \ldots \parallel (u_{n+1}, v_{m+1})
  \equiv^{*} (u_{1}, v_{1}) \parallel (u_{2}, v_{2}) \parallel \ldots \parallel (u_{n+1}, v_{m+1})
- t = u \pi_{i}. By induction hypothesis,
  u \equiv^{*} u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1}
  and each term u_{i}, for 1 \leq i \leq n + 1, is a simply typed \lambda-term. Applying several permutations we obtain
  t \equiv^{*} u_{1} \pi_{i} \parallel u_{2} \pi_{i} \parallel \ldots \parallel u_{n+1} \pi_{i}
  \equiv^{*} u_{1} \pi_{i} \parallel u_{2} \pi_{i} \parallel \ldots \parallel u_{n+1} \pi_{i}
- t = u \text{efq}_{p}. By induction hypothesis,
  u \equiv^{*} u_{1} \parallel u_{2} \parallel \ldots \parallel u_{n+1}
  and each term u_{i}, for 1 \leq i \leq n + 1, is a simply typed \lambda-term. Applying several permutations we obtain
  t \equiv^{*} u_{1} \text{efq}_{p} \parallel u_{2} \text{efq}_{p} \parallel \ldots \parallel u_{n+1} \text{efq}_{p}

The following, easy lemma shows that the activation phase of our reduction strategy is finite.

Lemma A.3 (Activate!). Let t be any term in parallel form that does not contain intuitionistic redexes and whose communication redexes have complexity at most \tau. Then there exists a finite sequence of activation reductions that results in a term \tau' that contains no redexes, except cross reduction redexes of complexity at most \tau.

Proof. The proof is by induction on the number n of subterms of the form \text{d}(u_{1} \parallel \ldots \parallel u_{m}) of t which are not active sessions. If there are no activation redexes in t, the statement trivially holds. Assume there is at least one activation redex \text{r} = \text{d}(u_{1} \parallel \ldots \parallel u_{m}). We apply an activation reduction to \text{r} and obtain a term \tau' with n − 1 subterms of the form \text{d}(u_{1} \parallel \ldots \parallel u_{m}) which are not active sessions. By the induction hypothesis to \tau', which immediately yields the thesis, we are left to verify that all communication redexes of \tau' have complexity at most \tau.

For this purpose, let \sigma be any channel variable which is bound in \tau'. Since \tau' is obtained from \tau just by renaming the non-active bound channel variable \alpha to an active one \alpha, every occurrence of \alpha in \tau' is of the form (ct)(\alpha/a) for some subterm ct of \tau. Thus ct(\alpha/a) = \alpha(\alpha/a)(t_{1}[\alpha/a], \ldots, t_{n}[\alpha/a]), where each t_{i} is a non-active subterm. It is enough to show that the value complexity of t_{1}[\alpha/a] is exactly the value complexity of t_{i}. We proceed by induction on the size of t_{i}. We can write t_{1} = r \sigma, where \sigma is a case-free stack. If \tau is of the form \text{Ax}w, (q_{1}, q_{2}), u(w), x, dw, with d channel variable, then the value complexity of t_{1}[\alpha/a] is the same as that of t_{i} (note if r = (q_{1}, q_{2}), then \sigma is not empty). If \tau = v_{0}[x_{1}, v_{1}, x_{2}, v_{2}], then \sigma is empty, otherwise s would contain a permutation redex. Therefore, the value complexity of t_{1}[\alpha/a] is the maximum among the value complexities of v_{1}[\alpha/a] and v_{2}[\alpha/a]. By induction hypothesis, their value complexities are respectively those of v_{1} and v_{2}, hence the value complexity of t_{1}[\alpha/a] is the same as that of t_{i}, which concludes the proof.

□

We shall need a simple property of the value complexity notion.

Lemma A.4 (Why Not 0). Let u be any simply typed \lambda-term and \sigma be a non-empty case-free stack. Then the value complexity of u\sigma is 0.

Proof. By induction on the size of u.

- If \sigma is of the form (\lambda x.w)\rho, t_{1}(w)\rho, (v_{1}, \rho)\rho, with case-free, then \sigma\rho has value complexity 0.
- If \sigma is of the form w\text{efq}_{p}, then \rho is atomic, thus \sigma must be empty, contrary to the assumptions.
- If \sigma is of the form \text{efq}_{p}[x_{1}, v_{1}, x_{2}, v_{2}]\rho, with \rho case-free, then by induction hypothesis the value complexities of v_{1}\rho\sigma and v_{2}\rho\sigma are 0 and since the value complexity of u\sigma is the maximum among them, u\sigma has value complexity 0.

□

As usual, in order to formally study redex contraction, we must consider simple substitutions that just replace some occurrences of a term with another, allowing capture of variables. In practice, it will always be clear from the context which of these occurrences will be replaced.

Definition A.8 (Simple Replacement). By s{t/u} we denote a term obtained from s replacing some occurrences of a term u with a term t of the same type of u, possibly causing capture of variables.
We now show an important property of the value complexity notion: the value complexity of \( w[v/s] \) either remains at most as it was before the substitution or becomes exactly the value complexity of \( v \).

**Lemma A.5** (The Change of Value). Let \( w, s, v \) be simply typed \( \lambda \)-terms with value complexity respectively \( \tau, \theta, \tau' \). Then the value complexity of \( w[v/s] \) is either at most \( \theta \) or equal to \( \tau' \). Moreover, if \( \tau' \leq \tau \), then the value complexity of \( w[v/s] \) is at most \( \theta \).

**Proof.** By induction on the size of \( w \) and by cases according to its possible shapes.

- \( w[v/s] = x[v/s] \). We have two cases.
  - \( s = x \). Then the value complexity of \( w[v/s] = v \) is \( \tau' \).
  - \( s \neq x \). The value complexity of \( w[v/s] = \theta \) and we are done.

- \( w[v/s] = \lambda xu[v/s] \). We have two cases.
  - \( \lambda xu[v/s] = \lambda xu(u[v/s]) \). Then the value complexity of \( w[v/s] = \theta \) and we are done.
  - \( \lambda xu[v/s] = \emptyset \). Then the value complexity of \( w[v/s] \) is \( \tau' \). Moreover, if \( \tau' \leq \tau \), since \( w = \lambda xu = s \), we have \( \theta = \tau \), thus \( \tau' \leq \theta \).

- \( w[v/s] = (q_1, q_2)(v[s]) \). We have two cases.
  - \( (q_1, q_2)(v[s]) = (q_1(v[s], q_2(v[s])) \). Then by induction hypothesis, the value complexities of \( q_1(v[s]) \) and \( q_2(v[s]) \) are at most \( \theta \) or equal to \( \tau' \). Since the value complexity of \( w[v/s] \) is the maximum among the value complexities of \( q_1(v[s]) \) and \( q_2(v[s]) \), we are done. Moreover, if \( \tau' \leq \tau \), by induction hypothesis, the value complexities of \( q_1(v[s]) \) and \( q_2(v[s]) \) are at most \( \theta \). Hence the value complexity of \( w[v/s] \) is at most \( \theta \) and we are done again.
  - \( (q_1, q_2)(v[s]) = \emptyset \). Then the value complexity of \( w[v/s] \) is \( \tau' \). Moreover, if \( \tau' \leq \tau \), since \( w = (q_1, q_2) = s \), we have \( \theta = \tau \), thus \( \tau' \leq \theta \).

- \( w[v/s] = \iota(u)(v[s]) \). We have two cases.
  - \( \iota(u)(v[s]) = \emptyset \). Then the value complexity of \( w[v/s] = \theta \) and we are done.
  - \( \iota(u)(v[s]) = u \). Then the value complexity of \( w[v/s] \) is \( \tau' \). Moreover, if \( \tau' \leq \tau \), since \( w = \iota(u) = s \), we have \( \theta = \tau \), thus \( \tau' \leq \theta \).

- \( w[v/s] = (v_0 z_1, v_1, z_2, v_2)(\rho[v/s]) \), where \( \rho \) is a case-free stack. If \( \rho \) is not empty, then by Lemma 5.4, the value complexity of \( w[v/s] \) is at most \( \theta \) or equal to \( \tau' \) and we are done. Moreover, if \( \tau' \leq \tau \), then by induction hypothesis, the value complexity of \( v_i[v/s] \) for \( i \in [1, 2] \) is at most \( \theta \). Hence, the value complexity of \( (v_0 z_1, v_1, z_2, v_2)(\rho[v/s]) \) is at most \( \theta \) and we are done. If \( \rho = \emptyset \), then \( \rho \) is a case-free stack and \( 1 \leq j \leq n \). If \( \rho = \emptyset \), then by Lemma 5.4, the value complexity of \( w[v/s] \) is \( \emptyset \) or \( \theta \), and we are done. So assume \( \rho = \emptyset \). Then the value complexity of \( w[v/s] \) is \( \tau' \). Moreover, if \( \tau' \leq \tau \), since it must be

The following lemma has the aim of studying mostly message passing and contraction of application and projection redexes.

**Lemma A.6** (Replace!). Let \( u \) be a term in parallel form, \( v, s \) be any simply typed \( \lambda \)-terms, \( \tau \) be the value complexity of \( v \) and \( \tau' \) be the maximum among the complexities of the channel occurrences in \( v \). Then every redex in \( u[v/s] \) is either (i) already in \( v \), (ii) of the form \( r[v/s] \) and has complexity smaller than or equal to the complexity of some redex \( r \) of \( u \), or (iii) has complexity \( \tau \) or \( \tau' \) or is a communication redex of complexity at most \( \tau' \).

**Proof.** We prove the following stronger statement.

- (i) Every redex and channel occurrence in \( u[v/s] \) is either (i) already in \( v \), (ii) of the form \( r[v/s] \) or \( aw[v/s] \) and has complexity smaller than or equal to the complexity of some redex in \( u \) or channel occurrence in \( u \), or (iii) has complexity \( \tau \) or \( \tau' \) or is a communication redex of complexity at most \( \tau' \).

We reason by induction on the size and by cases on the possible shapes of the term \( u \).

- \( (\lambda x t)\sigma[v/s] \), where \( \sigma = \sigma_1 \ldots \sigma_n \) is any case-free stack. By induction hypothesis, (i) holds for \( t[v/s] \) and \( \sigma_1[v/s] \) where \( 1 \leq i \leq n \). If \( (\lambda x t)\sigma(v[s]) = (\lambda x t)v(s) \), all the redexes and channel occurrences that we have to check are either in \( t[v/s] \), \( \sigma_1[v/s] \) or possibly, the head redex, thus the thesis holds. If \( (\lambda x t)\sigma(v[s]) = v(\sigma_1[v/s]) \ldots \sigma_n[v/s] \), then \( v(\sigma_1[v/s]) \) is a new intuitionistic redex, when \( v = ly \) and \( \sigma_1 = w_0[y_1, w_1, y_2, w_2] \). But the complexity of such a redex is equal to \( \tau \).

- \( (t_1,t_2)\sigma[v/s] \), where \( \sigma = \sigma_1 \ldots \sigma_n \) is any case-free stack. By induction hypothesis, (i) holds for \( t_1[v/s] \) and \( \sigma_1[v/s] \) where \( 1 \leq i \leq n \). If \( (t_1,t_2)\sigma(v[s]) = \sigma_1[v/s] \ldots \sigma_n[v/s] \), then \( \sigma_1[v/s] \) is a new intuitionistic redex, when \( v = ly \) and \( \sigma_1 = w_0[y_1, y_2, w_2] \). But the complexity of such a redex is equal to \( \tau \).
Disjunctive Axioms and Concurrent \(\lambda\)-Calculi: a Curry-Howard Approach

Here we study what happens after contracting an injection redex.

- \(\alpha\) occurring in some maximal complexity of the channel occurrences of the form \(\tau\) for \(t\), then \(\nu(\sigma_1(t)/\tau)\) could be a new intuitionistic redex, when \(\nu = \lambda w.\nu = \langle w_1, w_2, \nu \rangle = \nu = w_0[y_1, w_1, y_2, w_2]\). But the complexity of such a redex is equal to \(\tau\).
- \(\nu_0[z_1, w_1, z_2, w_2](\sigma(t)/v)\), where \(\sigma\) is any case-free stack. By induction hypothesis, \((*)\) holds for \(\nu_0(t)/v\), \(\nu_1(t)/v\), \(\nu_2(t)/v\) and \(\sigma_1(t)/v\) for \(1 \leq i \leq n\). If
  \[
  \nu_0[z_1, w_1, z_2, w_2](\sigma(t)/v) = \nu_0(t)/v[z_1, w_1(t)/v, z_2, w_2(t)/v](\rho(t)/v)
  \]
  we first observe that by Lemma 5.5, the value complexity of \(\nu_0(t)/v\) is at most that of \(\nu_0\) or exactly \(\tau\), therefore the possible injection or case permutation redex
  \[
  \nu_0(t)/v[z_1, w_1(t)/v, z_2, w_2(t)/v]
  \]
  satisfies the thesis. Again by Lemma 5.5, the value complexities of \(\nu_1(t)/v\) and \(\nu_2(t)/v\) are respectively at most that of \(\nu_1\) and \(\nu_2\) or exactly \(\tau\). Therefore the complexity of the possible case permutation redex
  \[
  \nu_0(t)/v(z_1, w_1(t)/v, z_2, w_2(t)/v)
  \]
is either \(\tau\), and we are done, or at most the value complexity of one among \(\nu_1\) and \(\nu_2\) thus at most the value complexity of the case permutation redex \(\nu_0(t)/v(z_1, w_1, z_2, w_2)\) and we are done.

- \(x \sigma(t)/v\), where \(x\) is any simply typed variable and \(\sigma = \sigma_1 \ldots \sigma_n\) is any case-free stack. By induction hypothesis, \((*)\) holds for \(\nu(t)/v\), \(\nu_1(t)/v\), \(\nu_2(t)/v\) and \(\sigma_1(t)/v\) for \(1 \leq i \leq n\). If \(x \sigma(t)/v = x(\sigma(t)/v)\), then there could be a new intuitionistic redex, when \(\nu = \lambda y.\nu = \langle q_1, q_2, \nu = \nu_i(q) = v = q_0[y_1, q_1, y_2, q_2]\). But the complexity of such a redex is \(\tau\).
- \(a \sigma(t)/v\), where \(a\) is a channel variable, \(t\) a term and \(\sigma = \sigma_1 \ldots \sigma_n\) is any case-free stack. By induction hypothesis, \((*)\) holds for \(t(v)/v, \sigma(t)/v\) \(1 \leq i \leq n\). If \(a \sigma(t)/v = \sigma(t_1)/v\), then \(\sigma(\nu(t)/v)\) could be an intuitionistic redex, when \(\nu = \lambda y.\nu = \langle w_1, w_2, \nu = \nu_i(w) = v = w_0[y_1, w_1, y_2, w_2]\). But the complexity of such a redex is equal to \(\tau\).
- \(a(t/v)(\sigma(t)/v)\), in order to verify the thesis it is enough to check the complexity of the channel occurrence \(a(t/v)\). By Lemma 5.5, the value complexity of \(t(v)/v\) is at most the value complexity of \(t\) or exactly \(\tau\), thus \((\alpha)\) or \((\alpha)\) holds.
- \(df(t_1 \ldots t_m)/v\). By induction hypothesis, \((*)\) holds for \(t_1(v)/v\) where \(1 \leq i \leq m\). The only redex in \(df(t_1 \ldots t_m)/v\) and not in some \(t_1(v)/v\) can be \(df(t_1(v)/v) \ldots t_m(v)/v\) itself. But the complexity of such redex equals the maximal complexity of the channel occurrences of the form \(aw\) occurring in some \(t_1(v)/v\), hence it is \(\tau\), at most \(\tau'\) or equal to the complexity of \(df(t_1 \ldots t_m)\).

\[
\text{Proof.}
\]

Let \(\nu = \nu_i(t)/x_1\) and \(s = \nu_i(t)/x_1, w_1, x_2, w_2\). We prove a stronger statement:

\((*)\) Any redex \(r\) in \(u(t)/v\) of complexity \(\theta\), either \(\nu_i(t)[x_1, w_1, x_2, w_2]\) has complexity greater than \(\theta\); or there is a redex in \(u\) of complexity \(\theta\) which belongs to the same group as \(r\) or is a case permutation redex. Moreover, for any channel occurrence in \(\nu(t)/v\) with complexity \(\theta'\), either \(\nu_i(t)[x_1, w_1, x_2, w_2]\) has complexity greater than \(\theta'\), or there is an occurrence of the same channel complexity with greater complexity or equal than \(\theta'\).

The proof is by induction on the size of \(u\) and by cases according to the possible shapes of \(u\):

- \((\alpha x t')\sigma(t)/v\), where \(\sigma = \sigma_1 \ldots \sigma_n\) is any case-free stack. By induction hypothesis, \((*)\) holds for \(t'/v\) and \(\sigma(t)/v\) for \(1 \leq i \leq n\). If \((\alpha x t')\sigma(t)/v = (\alpha x t')\sigma(t)/v\), all the redexes and channel occurrences that we have to check are either in \(\nu(t)/v\) or, possibly, the head redex, thus the thesis holds.
- \((t_1, t_2)\sigma(t)/v\), where \(\sigma = \sigma_1 \ldots \sigma_n\) is any case-free stack. By induction hypothesis, \((*)\) holds for \(t_1(v)/v, t_2(v)/v\) and \(\sigma_1(v)/v\) for \(1 \leq i \leq n\). If \((t_1, t_2)\sigma(t)/v = (t_1(v)/v, t_2(v)/v)\), all the redexes and channel occurrences that we have to check are either in \(\nu(t)/v\) or, possibly, the head redex, thus the thesis.

Here we study what happens after contracting an injection redex.
its redexes and channel occurrences are in $\sigma\{v/s\}$, thus the thesis holds. The case in which $x \sigma\{v/s\} = \nu(\sigma_1\{v/s\}) \ldots (\sigma_n\{v/s\})$ is impossible due to the form of $s = i(t)[x_1, x_2, x_3, w_1]$. 

\bullet \; v_0[z_1, v_1, z_2, v_2]\sigma\{v/s\}$, where $\sigma$ is any case-free stack. By induction hypothesis, (*) holds for $v_0[v/s], v_1[v/s], v_2[v/s]$ and $\sigma_i[v/s]$ for $1 \leq i \leq n$. If 

$$v_0[z_1, v_1, z_2, v_2]|\sigma\{v/s\} = v_0[z_1, v_1, v_2, v_2](\rho(v/s))$$

By Lemma 5.5 the value complexity of $w_0[t/x_1]$ is either at most the value complexity of $w_i$ or exactly the value complexity of $t$. In the first case, the value complexity of $w_0[t/x_1]$ is at most the value complexity of $w_i$ which is at most the value complexity of $\iota(t)[x_1, x_2, w_2]$. Thus, by Lemma 5.5 the value complexity of $t'(v/s)$ is at most the value complexity of $t'$ and we are done. In the second case, the value complexity of $t'(v/s)$ is the value complexity of $t$, which by Proposition 5.1 is at most the complexity of the type of $t$, thus smaller than the complexity of the injection redex $\iota(t)[x_1, x_2, w_2]$ occurring in $u$, which is what we wanted to show.

\bullet \; \theta(t_1 \parallel \ldots \parallel t_m)(v/s)$. By induction hypothesis, (*) holds for $t_i(v/s)$ for $1 \leq i \leq m$. The only redex in $\theta(t_1 \parallel \ldots \parallel t_m)(v/s)$ and not in some $t_i(v/s)$ can be $\theta(t_1[v/s]) \parallel \ldots \parallel t_{m}(v/s)$ itself. But the complexity of such redex equals the maximal complexity of the occurrences of the channel $a$ in the $t_i[v/s]$. Hence the statement follows.

\[\square\]

Here we study what happens after contracting a case permutation redex.

**Lemma A.8 (In Case!).** Let $u$ be a term in parallel form. Then for any redex $r_1$ of Group 1 in $u[t[x_1, v_1\xi, x_2, v_2\nu]/t[x_1, v_1, x_2, v_2\nu]]$, there is a redex in $u$ with greater or equal complexity than $r_1$; for any redex $r_2$ of Group 2 in $u[t[x_1, v_1\xi, x_2, v_2\nu]/t[x_1, v_1, x_2, v_2\nu]]$, there is a redex of Group 2 in $u$ with greater or equal complexity than $r_2$.

Proof. Let $t = t[x_1, v_1\xi, x_2, v_2\nu]$ and $s = t[x_1, v_1, x_2, v_2\nu]$. We prove the following stronger statement.

(*) For any Group 1 redex $r_1$ in $u[t[x_1, v_1\xi, x_2, v_2\nu]/t[x_1, v_1, x_2, v_2\nu]]$, there is a redex in $u$ with greater or equal complexity than $r_1$; for any Group 2 redex $r_2$ in $u[t[x_1, v_1\xi, x_2, v_2\nu]/t[x_1, v_1, x_2, v_2\nu]]$, there is a Group 2 redex in $u$ with greater or equal complexity than $r_2$.

Moreover, for any channel occurrence in $u[v/s]$ with complexity $0^\prime$, there is in $u$ an occurrence of the same channel with complexity greater or equal than $0^\prime$.

We first observe that the possible Group 1 redexes $v_1\xi$ and $v_2\nu$ have at most the complexity of the case permutation $t[x_1, v_1, x_2, v_2\nu]$. The rest of the proof is by induction on the shape of $u$.

\bullet \; (\lambda x t')\sigma\{v/s\}$, where $\sigma = \sigma_1 \ldots \sigma_n$ is any case-free stack. By induction hypothesis, (*) holds for $t'\nu/s$ and $\sigma_i\{v/s\}$ where $1 \leq i \leq n$. If $t'(v/s)$ is a case-free stack, all the redexes and channel occurrences that we have to check are either in $\sigma\{v/s\}$ or, possibly, the head redex, thus the thesis holds. There are no more cases since $\sigma$ is case-free.

\bullet \; (t_1, t_2)\sigma\{v/s\}$, where $\sigma = \sigma_1 \ldots \sigma_n$ is any case-free stack. By induction hypothesis, (*) holds for $t_1\{v/s\}, t_2\{v/s\}$ and $\sigma_i\{v/s\}$ where $1 \leq i \leq n$. If 

$$(t_1, t_2)\sigma\{v/s\} = (t_1\{v/s\}, t_2\{v/s\})(\sigma\{v/s\})$$

all the redexes and channel occurrences that we have to check are either in $\sigma\{v/s\}$ or, possibly, the head redex. The former are dealt with using the inductive hypothesis. As for the latter, it is immediate to see that the value complexity of $t[x_1, v_1\xi, x_2, v_2\nu]$ is equal to the value complexity of $t[x_1, v_1, x_2, v_2\nu]$. By Lemma 5.5, the value complexity of $t_i(v/s)$ is at most that of $t_i$, and we are done. There are no more cases since $\sigma$ is case-free.
• \( t_i(t') \sigma(v/s) \), where \( \sigma = \sigma_1 \ldots \sigma_n \) is any case-free stack. By induction hypothesis, \( (*) \) holds for \( t'_i(v/s) \) and \( \sigma_i(v/s) \) where \( 1 \leq i \leq n \). If \( t_i(t') \sigma(v/s) = t_i(t'_i/s_i)(\sigma(v/s)) \), all the redexes and channel occurrences that we have to check are either in \( \sigma(v/s) \) or, possibly, the head redex, thus the thesis holds. There are no more cases since \( \sigma \) is case-free.

• \( w_0(z_1, w_1, z_2, w_2) \sigma(v/s) \), where \( \sigma \) is any case-free stack. By induction hypothesis, \( (*) \) holds for \( w_0(v/s), v_1(v/s), v_2(v/s) \) and \( \sigma_1(v/s) \) for \( 1 \leq i \leq n \). If

\[
w_0(z_1, w_1, z_2, w_2) \sigma(v/s) = w_0(v/s)[z_1, w_1(v/s), z_2, w_2(v/s)](\sigma(v/s))
\]

Since the value complexity of \( t(x_1, v_1, x_2, v_2) \) is equal to the value complexity of \( t(x_1, v_1, x_2, v_2) \) by Lemma 5.5 the value complexities of \( w_0(v/s), w_1(v/s) \) and \( w_2(v/s) \) are respectively at most that of \( w_0, w_1 \) and \( w_2 \). Therefore the complexity of the possible case permutation redex

\[
(w_0(v/s)[z_1, w_1(v/s), z_2, w_2(v/s)](\sigma(v/s))\]

has complexity equal to the value complexity of \( w_0 \) and \( w_1 \) and we are done.

If \( w_0(z_1, w_1, z_2, w_2) \sigma(v/s) = v(\sigma_1(v/s)) \ldots (\sigma_n(v/s)) \), then there could be a new case permutation redex, because \( v = t(x_1, v_1, x_2, v_2) \). If \( \xi \) is free, by Lemma 5.4, this redex has complexity 0 and we are done; if not, it has the same complexity as \( t(x_1, v_1, x_2, v_2) \) and we are done.

• \( x \sigma(v/s) \), where \( x \) is any simply typed variable and \( \sigma = \sigma_1 \ldots \sigma_n \) is any case-free stack. By induction hypothesis, \( (*) \) holds for \( \sigma_i(v/s) \) where \( 1 \leq i \leq n \). If \( x \sigma(v/s) = x(\sigma(v/s)) \), all its redexes and channel occurrences are in \( \sigma(v/s) \), thus the thesis holds. There are no more cases since \( \sigma \) is case-free.

• \( a t \sigma(v/s) \), where \( a \) is a channel variable, \( t \) a term and \( \sigma = \sigma_1 \ldots \sigma_n \) is any case-free stack. By induction hypothesis, \( (*) \) holds for \( t \) and \( \sigma_i(v/s) \) where \( 1 \leq i \leq n \).

If \( a t \sigma(v/s) = a(t(v/s))\sigma(v/s) \), in order to verify the thesis it is enough to check the complexity of the channel occurrence \( a(t(v/s)) \). Since the value complexity of \( t(x_1, v_1, x_2, v_2) \) is equal to the value complexity of \( t(x_1, v_1, x_2, v_2) \), by Lemma 5.5 the value complexity of \( t(v/s) \) is at most that of \( t \), and we are done. There are no more cases since \( \sigma \) is case-free.

• \( d(t_1 \ldots \ldots t_m)(v/s) \). By induction hypothesis, \( (*) \) holds for \( t_i(v/s) \) where \( 1 \leq i \leq m \). The only redex in \( d(t_1 \ldots \ldots t_m)(v/s) \) and not in some \( t_i(v/s) \) can be \( d(t_1(v/s) \ldots \ldots t_m(v/s)) \) itself. But the complexity of such redex equals the maximal complexity of the occurrences of the channel \( a \) in \( t_i(v/s) \). Hence the statement follows.

\[ \Box \]

The following result is meant to break the possible loop between the intuitionistic phase and communication phase of our normalization strategy. Intuitively, when Group 1 redexes generate new redexes, these latter have smaller complexity than the former; when Group 2 redexes generate new redexes, these latter do not have worse complexity than the former.

**Proposition A.9 (Decrease!).** Let \( t \) be a term in parallel form, \( r \) be one of its redexes of complexity \( \tau \), and \( t' \) be the term that we obtain from \( t \) by contracting \( r \).

1. If \( r \) is a redex of the Group 1, then the complexity of each redex in \( t' \) is at most the complexity of a redex of the same group and occurring in \( t \); or is at most the complexity of a case permutation redex occurring in \( t \) or is smaller than \( \tau \).

2. If \( r \) is a redex of the Group 2 and not an activation redex, then every redex in \( t' \) either has the complexity of a redex of the same group occurring in \( t \) or has complexity at most \( \tau \).

**Proof.**

1) Suppose \( r = (\lambda x^A) v \), that \( s : B \) and let \( q \) be a redex in \( t' \) having different complexity from the one of any redex of the same group or the one of any case permutation occurring in \( t \). Since \( v : A \), we apply Lemma 5.6 to the term \( s(v/x^A) \). We know that if \( q \) occurs in \( s(v/x^A) \), since \( i \) and \( ii \) do not apply, it has the value complexity of \( v \), which by Proposition 5.1 is at most the complexity of \( A \), which is strictly smaller than the complexity of \( A \rightarrow B \) and thus than the complexity \( \tau \) of \( r \). Assume therefore that \( q \) does not occur in \( s(v/x^A) \). Since \( s(v/x^A) : B \), by applying Lemma 5.6 to the term \( t = (t(s/x^A)/A^2(v), s) \) we know that \( q \) has the same complexity as the value complexity of \( s(v/x^A) \) – which by Proposition 5.1 is at most the complexity of \( B \) – or is a communication redex of complexity equal to the complexity of some channel occurrence \( a(w/v/x^A) \) in \( s(v/x^A) \), which by Lemma 5.5 is at most the complexity of \( A \) or at most the complexity of \( aw \), and we are done again.

Suppose that \( r = (u(s)[x^1, u_1, y^B, u_2]) \). By applying Lemma 5.7 to

\[ t' = t[u(s)[x^1, u_1, y^B, u_2]] \]

we are done.

2) Suppose \( r = (\langle v_0, v_1 \rangle)_{i \in I} \), \( \langle v_0, v_1 \rangle : A_0 \wedge A_1 \). Let \( q \) be a redex in \( t' \) having greater complexity than that of any redex of the same group in \( t \). The term \( q \) cannot occur in \( v_i \), by the assumption just made. Moreover, by Proposition 5.1, the value complexity of \( v_i \) cannot be greater than the complexity of \( A_j \). By applying Lemma 5.6 to the term \( t = t(v_i/r) \), we know that \( q \) has complexity equal to the value complexity of \( v_i \), because by the assumption on \( q \) the cases \( i \) and \( ii \) of Lemma 5.6 do not apply. Such complexity is at most the complexity \( \tau \) of \( r \). Thus we are done.

Suppose that \( r = s(x^A, u_1, y^B, v) \) is a case permutation redex. By applying Lemma 5.8 we are done.

If \( t' \) is obtained by performing a communication permutation, then obviously the thesis holds.

If \( t' \) is obtained by a cross reduction of the form \( d(u_1 \ldots \ldots u_m) \leftrightarrow u_{j_1} \ldots \ldots u_{j_m}, \) for \( 1 \leq j_1 < \ldots < j_m \leq m, \) then there is nothing to prove: all redexes occurring in \( t' \) also occur in \( t \).

Suppose now that

\[ r = d(G_1[a_{F_1}^{G_1} t_1] \ldots \ldots [C_m^{G_1} a_{F_m}^{G_1} t_m]) \]

\[ t_1 = (u_1 \ldots \ldots u_{j_1}) \). Then by cross reduction, \( r \) reduces to \( y(sl_1 \ldots \ldots t_m) \) where if \( G_i \neq \bot \):

\[ s_l = d(G_1[a_{F_1}^{G_1} t_1] \ldots \ldots [C_i^{G_1} a_{F_i}^{G_1} y(sl_1)] \ldots \ldots [C_m^{G_1} a_{F_m}^{G_1} t_m]) \]

and if \( G_i = \bot \):

\[ s_l = d(G_1[a_{F_1}^{G_1} t_1] \ldots \ldots [C_i^{G_1} a_{F_i}^{G_1} y(sl_1)] \ldots \ldots [C_m^{G_1} a_{F_m}^{G_1} t_m]) \]
in which \( F_i = G_i \) and \( t' = t(r'/r) \). Let now \( q \) be a redex in \( t' \) having different complexity from the one of any redex of the same
group in $t$. We first show that it cannot be an intuitionistic redex: assume it is. Then it occurs in one of the terms $C_i t_j^{b_i(y_j/y_j)}$ or $C_i [b_i(y_j)]$. By applying Lemma 5.6 to them, we obtain that the complexity of $q$ is the value complexity of $t_j^{b_i(y_j/y_j)}$ or $b_i(y_j)$, which, by several applications of Lemma 5.5, are at most the value complexity of $t_j$ or 0 and thus by definition at most the complexity of $r$, which is a contradiction. Assume therefore that $g = \{ s_1 \| \cdots \| s_p \}$ is a communication redex. Every channel occurrence of $c$ in the terms $C_i t_j^{b_i(y_j/y_j)}$ or $C_i [b_i(y_j)]$ is of the form $c w(t_j^{b_i(y_j/y_j)}/a t_k)$ or $c w(b_i(y_j)/a t_k)$ where $c w$ is a channel occurrence in $t$. By Lemma 5.5, each of these occurrences has either at most the value complexity of $c w$ or has at most the value complexity of $r$, which is a contradiction. □

The following result is meant to break the possible loop during the communication phase: no new activation is generated after a cross reduction, when there is none to start.

**Lemma A.10 (Freeze!).** Suppose that $s$ is a term in parallel form that does not contain projection nor case permutation nor activation redexes. Let $d(q_1 \| \cdots \| q_m)$ be some redex of $s$ of complexity $r$. If $s'$ is obtained from $s$ by performing first a cross reduction on $d(q_1 \| \cdots \| q_m)$ and then contracting all projection and case permutation redexes, then $s'$ contains no activation redexes.

**Proof.** Let $d(q_1 \| \cdots \| q_m) = d(C_i[a^{F_i - G_i} t_i \sigma_i] \| \cdots \| G_m[a^{F_m - G_m} t_m \sigma_m])$ where $\sigma_i$ for $1 \leq i \leq m$ are the stacks which are applied to $a t_i$, and $t$ be the cross reduction redex occurring in $s$ that we reduce to obtain $s'$. Then after performing the cross reduction and contracting all the intuitionistic redexes, $t$ reduces to $d(s_1 \| \cdots \| s_m)$ where if $G_i \neq \bot$:

$$s_i = d(C_i[a^{F_i - G_i} t_i \sigma_i] \| \cdots \| G_m[a^{F_m - G_m} t_m \sigma_m])$$

and if $G_i = \bot$:

$$s_i = d(C_i[a^{G_i - G_i} t_i] \| \cdots \| G_m[a^{F_m - G_m} t_m])$$

in which $F_j = G_i$, where $t_i'$ are the terms obtained reducing all projection and case permutation redexes respectively in $t_j^{b_i(y_j/y_j)}$. Moreover $s' = s[t'/t]$.

We observe that $a$ is active and hence the terms $s_i$ for $1 \leq i \leq m$ are not activation redexes. Moreover, since all occurrences of $b$ are of the form $b_i(y_j)$, $t'$ is not an activation redex. Now we consider the channel occurrences in any term $s_i$. We first show that there are no active channels in $t_j'$ which are bound in $s'$. For any subterm $w$ of $t_j$, since for any stack $\theta$ of projections, $b_i(y_j)$ has value complexity $\theta$, we can apply repeatedly Lemma 5.5 to $w b_i(y_j)/y_j$ and obtain that there is no value complexity of $w b_i(y_j)/y_j$ is exactly the value complexity of $w$. This implies that there is no active channel in $t_j^{b_i(y_j/y_j)}$ because there is none in $t_j$. The following general statement immediately implies that there is no active channel in $t_j'$ which is bound in $s'$ either.

(*) Suppose that $r$ and $\theta$ are respectively a simply typed $\lambda$-term and a stack contained in $s'$ that do not contain projection and permutation redexes, nor active channels bound in $s'$. If $r'$ is obtained from $r \theta$ by performing all possible projection and case permutation reductions, then there are no active channels in $r'$ which are bound in $s'$.

Proof. By induction on the size of $r$. We proceed by cases according to the shape of $r$.

- If $r = \lambda \omega \cdot \tau = t_1(w)$, $r = \mathrm{we} q \rho$, $r = x$ or $r = aw$, for a channel $a$, then $r' = r \theta \rho$ and the thesis holds.
- If $r = \langle \tau_0, v_1 \rangle$ the only redex that can occur in $r \theta \rho$ is a projection redex, where $\theta = \{ \rho \}$. Therefore, $r \theta \rho \Rightarrow v_\theta \rho \Rightarrow r'$. By induction hypothesis applied to $v_\theta \rho$, there are no active channels in $r'$ which are bound in $s'$.
- If $r = t[x, v_0, v_1]$, then $r \theta \rho \Rightarrow t[x, v_0 \theta, y, v_1 \theta] \Rightarrow t[x, v_0, y, v_1] = r'$. By induction hypothesis applied to $v_0 \theta$, there are no active channels in $v_\theta$ and we are done.
- If $r = p v \xi$, with $\xi$ case free, then $r' = p v \xi$ and the thesis holds.

Now, let $c$ be any non-active channel bound in $s'$ occurring in $C_i[t_j']$ or $C_i[b_i(y_j)]$ but not in $u'$: any of its occurrences is of the form $c(p_1, \ldots, p_l(u'/avrp), \ldots, p_m)$, where each $p_i$ is not a pair. We want to show that the value complexity of $p_i(u'/avrp)$ is exactly the value complexity of $p_i$. Indeed, $p_i = r \nu$ where $\nu$ is a case-free stack. If $r$ is of the form $\lambda \omega \cdot \tau = t_1(w), x, dv$, with $d \neq a$, then the value complexity of $p_i(u'/avrp)$ is the same as that of $p_i$ (note that if $r = \langle q_1, q_1 \rangle$, then $\nu$ is not empty). If $r = t_{\nu_0}(x_1, v_1, x_2, v_2)$, then $\nu$ is empty, otherwise $w$ would contain a permutation redex, so $c(p_1, \ldots, p_l(u'/avrp), \ldots, p_m)$ is active, and there is an activation redex in $s$, which is contrary to our assumptions. The case $r = av, v = pp'$ is also impossible, otherwise $c(p_1, \ldots, p_l(u'/avrp), \ldots, p_m)$ would be active, and we are done. □

**Definition A.9.** We define the height $h(t)$ of a term $t$ in parallel form as

- $h(u) = 0$ if $u$ is simply typed $\lambda$-term
- $h(u)_{\langle \omega \nu \rangle} = 1 + \max(h(u), h(\nu))$

The communication phase of our reduction strategy is finite.

**Lemma A.11 (Communicate!).** Let $t$ be any term in parallel form that does not contain projection, case permutation, or activation redexes. Assume moreover that all redexes in $t$ have complexity at most $r$. Then $t$ reduces to a term containing no redexes, except Group 1 redexes of complexity at most $r$.

**Proof.** We prove the statement by lexicographic induction on the triple $(n, h, g)$ where

- $n$ is the number of subterms $d(u_1 \| \cdots \| u_m)$ of $t$ such that $d(u_1 \| \cdots \| u_m)$ is an active, but not uppermost, session.
- $h$ is the function mapping each natural number $m \geq 2$ into the number of uppermost active sessions in $t$ with height $m$. 
- $g$ is the function mapping each natural number $m$ into the number of uppermost active sessions $d(u_1 \| \cdots \| u_m)$ in $t$ containing $m$ occurrences of a $c$.

We employ the following lexicographic ordering between functions for the second and third elements of the triple: $f < f'$ if and only if there is some $i$ such that for all $j > i$, $f(j) = f'(j) = 0$ and there is some $i$ such that for all $j > i$, $f(j) = 0$ and $f(i) < f'(i)$.

If $(h, g) > 0$, for some $j \geq 2$, then there is at least an active session $d(u_1 \| \cdots \| u_m)$ in $t$ that does not contain any active session and such that $h(d(u_1 \| \cdots \| u_m)) = j$. Hence $u_j = \{ s_1 \| \cdots \| s_q \}$ for some $1 \leq i \leq m$. We obtain $t'$ by applying inside $t$ the permutation $d(u_1 \| \cdots \| u_j \| s_1 \| \cdots \| s_q) \cdots u_m) \Rightarrow d(u_1 \| \cdots \| s_1 \| \cdots \| s_q) \cdots u_m)$. We claim that the term $t'$ thus obtained has complexity has complexity $(n, h', g')$, with $h' < h$. Indeed, $d(u_1 \| \cdots \| u_m)$ does not contain active sessions,
thus $b$ is not active and the number of active sessions which are not uppermost in $t'$ is still $n$. With respect to $t$, the term $t'$ contains one less uppermost active session with height $j$ and $q$ more of height $j - 1$, and therefore $h' < h$. Furthermore, since the permutations do not change at all the purely intuitionistic subterms of $t$, no new activation or intuitionistic redex is created. In conclusion, we can apply the induction hypothesis on $t'$ and thus obtain the thesis.

If $h(m) = 0$ for all $m \geq 2$, then let us consider an uppermost active session $a(u_1 \ldots \ldots u_m)$ in $t$ such that $h(a(u_1 \ldots \ldots u_m)) = 1$; if there is not, we are done. We reason by cases on the distribution of the occurrences of $a$. Either (i) some $u_i$ for $1 \leq i < m$ does not contain any occurrence of $a$, or (ii) all $u_i$ for $1 \leq i \leq m$ contain some occurrence of $a$.

Suppose that (i) is the case and, without loss of generality, that $a$ occurs $j$ times in $u$ and does not occur in $v$. We then obtain a term $t'$ by applying a cross reduction $a(u_1 \ldots \ldots u_m) \mapsto u_j \ldots \ldots u_{j-1}$ where $u_i$ is an active session $a(u_1 \ldots \ldots u_n)$ in $t$ such that $h(a(u_1 \ldots \ldots u_n)) = 1$; if there is not, we are done. We reason by cases on the distribution of the occurrences of $a$. Either (i) some $u_i$ for $1 \leq i < m$ does not contain any occurrence of $a$, or (ii) all $u_i$ for $1 \leq i \leq m$ contain some occurrence of $a$.

Suppose now that (ii) is the case and all $u_i$ for $1 \leq i \leq m$ together contain $j$ occurrences of $a$. Then $a(u_1 \ldots \ldots u_m)$ is of the form $a(C_1[a_{F_1 \rightarrow G_1} t_1] \ldots \ldots C_m[a_{F_m \rightarrow G_m} t_m])$ where $a$ is active, $C_i[a_{F_i \rightarrow G_i} t_i]$ for $1 \leq i \leq m$ are simply typed $\lambda$-terms; $a_{F_i \rightarrow G_i}$ is rightmost in each of them. Then we can apply the cross reduction $a(C_1[a_{F_1 \rightarrow G_1} t_1] \ldots \ldots C_m[a_{F_m \rightarrow G_m} t_m]) \mapsto b(s_1 \ldots \ldots s_m)$ in which $b$ is fresh and for $1 \leq i \leq m$, we define, if $G_i = \alpha \not\vdash s_i = a(C_1[a_{F_1 \rightarrow G_1} t_1] \ldots \ldots C_i[b_{i}(y_i)] \ldots \ldots C_m[a_{F_m \rightarrow G_m} t_m])$ and if $G_i = \alpha \vdash s_i = a(C_1[a_{F_1 \rightarrow G_1} t_1] \ldots \ldots C_i[b_{i}(y_i)] \ldots \ldots C_m[a_{F_m \rightarrow G_m} t_m])$ where $F_i = G_i; y_i$ for $1 \leq i \leq m$ is the sequence of the free variables of $t_i$ bound in $C_i[a_{F_i \rightarrow G_i} t_i]; b_i = b_{i_{F_i \rightarrow G_i}} B_i$, where $B_i$ for $1 \leq i \leq m$ is the type of $(y_i)$. By Lemma 5.10, after performing all projections and case permutation reductions in all $C_i[b_i(y_i) A_i y_i]$ and $C_i[b_i(y_i)]$ for $1 \leq i \leq m$ we obtain a term $t'$ that contains no activation redexes; moreover, by Proposition 5.9.2., $t'$ contains only redexes having complexity at most $r$.

We claim that the term $t'$ thus obtained has complexity $\langle n, h, q \rangle$ where $q' < q$. Indeed, the value $n$ does not change because all newly introduced occurrences of $b$ are not active. The new active sessions $s_i = a(C_1[a_{F_1 \rightarrow G_1} t_1] \ldots \ldots C_i[b_{i}(y_i)] \ldots \ldots C_m[a_{F_m \rightarrow G_m} t_m])$ or $s_i = a(C_1[a_{F_1 \rightarrow G_1} t_1] \ldots \ldots C_i[b_{i}(y_i)] \ldots \ldots C_m[a_{F_m \rightarrow G_m} t_m])$ for $1 \leq i \leq m$ have all height $1$ and contain $j - 1$ occurrences of $a$. Since furthermore the reduced term does not contain channel occurrences of any uppermost active session different from $a(u_1 \ldots \ldots u_m)$ we can infer that $g'(j) = g(j) - 1$ and that, for any $i$ such that $i > j$, $g'(i) = g(i)$.

We can apply the induction hypothesis on $t'$ and obtain the thesis.

We now combine together all the main results achieved so far.

**Proposition A.12** (Normalize!). Let $t : A$ be any term in parallel form. Then $t \mapsto t'$, where $t'$ is a parallel normal form.

**Proof.** Let $r$ be the maximum among the complexity of the redexes in $t$. We prove the statement by induction on $r$.

Starting from $t$, we reduce all intuitionistic redexes and obtain a term $t_1$ that, by Proposition 5.9, does not contain redexes of complexity greater than $r$. By Lemma 5.3, $t_1 \mapsto t_2$ where $t_2$ does not contain any redex, except cross reduction redexes of complexity at most $r$. By Lemma 5.11, $t_2 \mapsto t_3$ where $t_3$ contains only Group 1 redexes of complexity at most $r$. Suppose $t_3 \mapsto t_4$ by reducing all Group 1 redexes, starting from $t_3$. By Proposition 5.9, every Group 1 redex which is generated in the process has complexity at most $r$, thus every Group 2 redex which is generated has complexity smaller than $r$, thus $t_4$ can only contain redexes with complexity smaller than $r$. By induction hypothesis $t_4 \mapsto t'$, with $t'$ in parallel normal form.

The normalization for $\lambda Ax$ now easily follows.

**Theorem A.13.** Suppose that $t : A$ is a proof term of $G$. Then $t \mapsto t' : A$, where $t'$ is a normal parallel form.

**B The Subformula Property**

We now show that normal $\lambda Ax$-terms satisfy the Subformula Property: a normal proof does not contain concepts that do not already appear in the premises or in the conclusion. This, in turn, implies that our Curry–Howard correspondence for $\lambda Ax$ is meaningful from the logical perspective and produces analytic proofs.

**Proposition B.1** (Parallel Normal Form Property). If $t \in NF$ is a $\lambda Ax$-term, then it is in parallel normal form.

**Proof.** By induction on $t$.

- $t$ is a variable $x$. Trivial.
- $t = \lambda x.v$. Since $t$ is normal, $v$ cannot be of the form $a(u_1 \ldots \ldots u_m)$, otherwise one could apply the permutation $t = \lambda x A \overset{\tau}{\rightarrow} a(\lambda x A u_1 \ldots \ldots u_m)$.
- $t = \alpha \not\vdash (\alpha)$. Hence, by induction hypothesis $\alpha \not\vdash (\alpha)$.
- $t = \langle v_1, v_2 \rangle$. Since $t$ is normal, neither $v_1$ nor $v_2$ can be of the form $a(u_1 \ldots \ldots u_m)$, otherwise one could apply one of the permutations $a(u_1 \ldots \ldots u_m) \mapsto a(u_1, v_1) \ldots \ldots a(u_m, v_2)$ and $t$ would not be in normal form. Hence, by induction hypothesis $v_2$ and $v_2$ must be simply typed $\lambda$-terms.
- $t = v_1, v_2$. Since $t$ is normal, neither $v_1$ nor $v_2$ can be of the form $a(u_1 \ldots \ldots u_m)$, otherwise one could apply one of the permutations $a(u_1 \ldots \ldots u_m) \mapsto a(u_1, v_1) \ldots \ldots a(u_m, v_2)$ and $t$ would not be in normal form. Hence, by induction hypothesis $v_1$ and $v_2$ must be simply typed $\lambda$-terms.
Theorem B.2 (Subformula Property). Suppose
\[ x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{D_1}, \ldots, a_m^{D_m} \vdash t : A \quad \text{and} \quad t \in \text{NF}. \]

Then:
1. For each channel variable \( a^{B \rightarrow C} \) occurring bound in \( t \), the prime factors of \( B, C \) are subformulas of \( A_1, \ldots, A_n, A \) or proper subformulas of \( D_1, \ldots, D_m \).
2. The type of any subterm of \( t \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) and of proper subformulas of \( D_1, \ldots, D_m \).

Proof. We proceed by structural induction on \( t \) and reason by cases, according to the form of \( t \):
- \( t = \langle u, v \rangle : F \land G \). Since \( t \in \text{NF} \), by Proposition 6.1 it is in parallel form, thus is a simply typed \( \lambda \)-term. Therefore no communication variable can be bound inside \( t \), thus 1. trivially holds. By induction hypothesis, 2. holds for \( u : F \) and \( v : G \). Therefore, the type of any subterm of \( u \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) and of proper subformulas of \( D_1, \ldots, D_m \) and any subterm of \( v \) is either a subformula or a conjunction of subformulas of some \( A_1, \ldots, A_n, A \) and \( F \) of proper subformulas of \( D_1, \ldots, D_m \). Therefore, any subformula of \( F \) and \( G \) must be a subformula of the type \( F \land G \) of \( t \). Hence the type of any subterm of \( \langle u, v \rangle \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) and \( F \land G \) or a proper subformula of \( D_1, \ldots, D_m \) and the statement holds for \( t \).
- \( t = \lambda x^F u : F \rightarrow G \). Since \( t \in \text{NF} \), by Proposition 6.1 it is in parallel form, thus is a simply typed \( \lambda \)-term. Therefore no communication variable can be bound inside \( t \), thus 1. trivially holds. By induction hypothesis, 2. holds for \( u : F \) and \( v : G \). Therefore the type of any subterm of \( u \) is either a subformula or a conjunction of subformulas of some \( A_1, \ldots, A_n, A \) and \( F \) of proper subformulas of \( D_1, \ldots, D_m \). Since the type \( F \) of \( x \) is a subformula of \( F \rightarrow G \), the type of any subterm of \( \lambda x^F u \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) and \( F \rightarrow G \) or a proper subformula of \( D_1, \ldots, D_m \) and the statement holds for \( t \).
- \( t = u_i(u) : F \lor G \) for \( i \in \{0, 1\} \). Without loss of generality assume that \( i = 1 \) and \( u : G \). Since \( t \in \text{NF} \), by Proposition 6.1 it is in parallel form, thus is a simply typed \( \lambda \)-term. Therefore no communication variable can be bound inside \( t \), thus 1. trivially holds. By induction hypothesis, 2. holds for \( u : F \).

Therefore, the type of any subterm of \( u \) is either a subformula or a conjunction of subformulas of some \( A_1, \ldots, A_n, A \) of \( F \) or proper subformulas of \( D_1, \ldots, D_m \). Moreover, any subformula of \( G \) must be a subformula of the type \( F \lor G \) of \( t \). Hence the type of any subterm of \( u_i(u) \) is either a subformula or a conjunction of subformulas of some \( A_1, \ldots, A_n, A \) and \( F \lor G \) or a proper subformula of \( D_1, \ldots, D_m \) and the statement holds for \( t \) as well.
- \( t = x^A \sigma : A \) for some \( A \) among \( A_1, \ldots, A_n \) and stack \( \sigma \). Since \( t \in \text{NF} \), in parallel form, thus is a simply typed \( \lambda \)-term and no communication variable can be bound inside \( t \). By induction hypothesis, for any element \( \sigma_j : S_j \) of \( \sigma \), the type of any subterm of \( \sigma_j \) is either a subformula or a conjunction of subformulas of some \( A_1, \ldots, A_n \) of the type \( S_j \) of \( \sigma_j \) and of proper subformulas of \( D_1, \ldots, D_m \).

If \( \sigma \) is case-free, then every \( S_j \) is a subformula of \( A_j \), or of \( A \), when \( \sigma = \sigma \cdot \text{nf} \cdot \text{A} \). Hence, the type of any subterm of \( x^A \sigma \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) or of proper subformulas of \( D_1, \ldots, D_m \) and the statement holds for \( t \) as well.

In case \( \sigma \) is not case-free, then, because of case permutations, \( \sigma = \sigma \cdot [y^G \cdot v_1, z^E \cdot v_2] \), with \( \sigma \cdot \text{case-free} \). By induction hypothesis we know that the type of any subterm of \( v_1 : A \) and \( v_2 : A \) is either a subformula or a conjunction of subformulas of some \( A_1, \ldots, A_n \), of \( A, G, E \) and of proper subformulas of \( D_1, \ldots, D_m \). Moreover, \( G \) and \( E \) are subformulas of \( A_1 \) due to the properties of stacks. Hence, the type of any subterm of \( x^A \sigma \cdot x^G \cdot y^{G_1}, z^{C_1} \cdot v_2 \) is either a subformula or a conjunction of subformulas of \( A_1, \ldots, A_n, A \) and of proper subformulas of \( D_1, \ldots, D_m \) and also in this case the statement holds for \( t \) as well.

- \( t = a^{D} u \sigma : A \) for some \( D \) among \( D_1, \ldots, D_n \) and stack \( \sigma \). As in the previous case.

- \( t = y^A u_1, \ldots, u_k : A \) and \( b^{D_1} \cdot H_1 \) occurs in \( u_i \). Suppose, for the sake of contradiction, that the statement does not hold. We know by induction hypothesis that the statement holds for \( u_i : A_1, \ldots, u_k : A \). We first show that it cannot be the case that

(+) all prime factors of \( G_1, H_1, \ldots, G_k, H_k \) are subformulas of \( A_1, \ldots, A_n, A \) or proper subformulas of \( D_1, \ldots, D_m \).

Indeed, assume by contradiction that (+) holds. Let us consider the type \( T \) of any subterm of \( t \) which is not a bound communication variable and the formulas \( B, C \) of any bound communication variable \( a^{B \rightarrow C} \) of \( t \). Let \( P \) be any prime factor of \( T \) or \( B \) or \( C \). By induction hypothesis applied to \( u_i, \ldots, u_n \), we obtain that \( P \) is either subformula or conjunction of subformulas of some \( A_1, \ldots, A_n \) and of proper subformulas of \( D_1, \ldots, D_m, G_1, H_1, \ldots, G_k, H_k \). Moreover, \( P \) is prime and so it must be subformula of \( A_1, \ldots, A_n, A \) or a proper subformula of \( D_1, \ldots, D_m \) or a prime factor of \( G_1, H_1, \ldots, G_k, H_k \). Since (+) holds, \( P \) must be a subformula of \( A_1, \ldots, A_n, A \) or proper subformula of \( D_1, \ldots, D_m \), and this contradicts the assumption that the subformula property does not hold for \( t \).

We shall say from now on that any bound channel variable \( a^{F_1 \rightarrow F_2} \) of \( t \) violates the subformula property maximally (due to \( Q \)) if (i) some prime factor \( Q \) of \( F_1 \) or \( F_2 \) is neither a subformula of \( A_1, \ldots, A_n, A \) nor a proper subformula of

Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco
Disjunctive Axioms and Concurrent λ-Calculi: a Curry-Howard Approach

Technical report, ArXiv, 2018

D₁, . . . , Dₘ and (ii) for every other bound channel variable cS₁→S₂ of t, if some prime factor Q of S₁ or S₂ is neither a subformula of A₁, . . . , Aₙ, A nor a proper subformula of D₁, . . . , Dₘ, then Q is complex at most as Q. If Q is a subformula of F₁, we say that dE₁→F₁ violates the subformula property maximally in the input.

It follows from (∗) that a channel variable maximally violating the subformula property must exist. We show now that there also exists a subterm cF₁→F₂ w of t such that c maximally violates the subformula property in the input due to Q, and w does not contain any channel variable that violates the subformula property maximally.

In order to prove the existence of such term, we prove

(∗∗) Let t₁ be any subterm of t such that t₁ contains at least a maximally violating channel and all maximally violating channel of t that are free in t₁ are maximally violating in the input. Then there is a simply typed subterm s of t₁ such that s contains at least a maximally violating channel, and such that all occurrences of maximally violating channels occurring in s violate the subformula property in the input.

We proceed by induction on the number n of ∥ operators that occur in t₁.

If n = 0, it is enough to pick s = t₁.
If n > 0, let t₁ = d(u₁ ∥ . . . ∥ uₙ) and assume dE₁→F₁ occurs in u₁. If no dE₁→F₁ maximally violates the subformula property, we obtain the thesis by applying the induction hypothesis to any u₁. Assume therefore that some dE₁→F₁ maximally violates the subformula property due to Q. Then there is some dE₂→F₂ such that Q is a prime factor of E₁ or E₂. By induction hypothesis applied to u₁ or u₂, we obtain the thesis.

By (∗∗) we can infer that in t there is a simply typed λ-term s that contains at least one occurrence of a maximally violating channel and only occurrences of maximally violating channels that are maximally violating in the input. The rightmost of the maximally violating channel occurrences in s is then of the form cF₁→F₂ w where c maximally violates the subformula property in the input and w does not contain any channel variable maximally violating the subformula property.

Consider now this term cF₁→F₂ w.

Since Q is a prime factor of F₁, it is either an atom P or a formula of the form Q' → Q'' or of the form Q' ∨ Q''.

Let w = (w₁, . . . , wₖ), where each wi is not a pair, and let k be such that Q occurs in the type of wₖ.

We start by ruling out the case that wₖ = λy s or wₖ = iₖ(s) for i ∈ {0, 1}, otherwise it would be possible to perform an activation reduction or a cross reduction to some subterm u' ∥ v', which must exist since c is bound.

Suppose now, by contradiction, that wₖ = xT σ where σ is a stack. It cannot be the case that σ = σ'[yF₁→F₂, zE₂→F₂] nor σ = σ'efqP, otherwise we could apply an activation or cross reduction. Therefore σ is case-free and does not contain efqP. Moreover, xT cannot be a free variable of t, then T = Aᵢₖ, for some 1 ≤ i ≤ n, and Q is a subformula of Aᵢₖ, contradicting the assumptions.

Suppose therefore that xT is a bound intuitionistic variable of t, such that t has a subterm λxT₁ : T → Y or, without loss of generality, s[xT₁,v₁,zE₂→F₂] with s : T ∨ Y for some formula Y. By induction hypothesis T → Y and T ∨ Y are subformulas of A₁, . . . , Aₙ, A or proper subformulas of D₁, . . . , Dₘ, G → H. But T → Y and T ∨ Y contain Q as a proper subformula and cF₁→F₂ w violates maximally the subformula due to Q. Therefore T → Y and T ∨ Y are neither subformulas of A₁, . . . , Aₙ, A nor proper subformulas of D₁, . . . , Dₘ and thus must be proper subformulas of G → H. Since cF₁→F₂ w violates the subformula property maximally due to Q, T → Y and T ∨ Y must be at most as complex as Q, which is a contradiction.

Suppose now that xT is a bound channel variable, thus wₖ = aT r σ, where aT is a bound communication variable of t, with T = T₁ → T₂. Since cF₁→F₂ w is rightmost, a ≠ c. Moreover, Q is a subformula of a prime factor of T₂, whereas aT₁→T₁ occurs in w, which is impossible by choice of c. This contradicts the assumption that the term is normal and ends the proof.

□