FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Intermediate logics and concurrent λ-calculi: A proof-theoretical approach

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Francesco Antonio Genco
Matrikelnummer 01428996

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Agata Ciabattoni
Zweitbetreuung: Federico Aschieri
und Ezio Bartocci

Diese Dissertation haben begutachtet:

Arnon Avron

Michel Parigot

Philip Wadler

Wien, 11. März 2019

Francesco Antonio Genco

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Intermediate logics and concurrent λ-calculi: A proof-theoretical approach

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Francesco Antonio Genco
Registration Number 01428996

to the Faculty of Informatics

at the TU Wien

Advisor: Agata Ciabattoni
Co-advisors: Federico Aschieri
and Ezio Bartocci

The dissertation has been reviewed by:

Arnon Avron

Michel Parigot

Philip Wadler

Vienna, 11th March, 2019

Francesco Antonio Genco

Erklärung zur Verfassung der Arbeit

Francesco Antonio Genco
Schönburgstraße 26, Top 23
1040 Wien
Österreich

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. März 2019

Francesco Antonio Genco

Acknowledgements

Kurzfassung

Der mögliche Zusammenhang des Hypersequentialkalküls in der Beweistheorie mit der Theorie der nebenläufigen Programmierung wurde 1991 von Avron vermutet. Er schlug vor, ein Hypersequent als parallele Verknüpfung mehrerer Sequente zu betrachten, und hielt es für möglich, die von den Hypersequentialkalkülen beschriebenen intermediären Logiken als Grundlage paralleler λ-Kalküle zu verwenden.

In der vorliegenden Dissertation entwickeln wir auf der Curry-Howard-Korrespondenz basierende parallele und nebenläufige λ-Kalküle für solche Logiken, die sich im Hypersequentialkalkül formalisieren lassen. Die erhaltenen Kalküle sind ausdrucksstärker als der einfach typisierte Lambda-Kalkül, und erlauben die Formalisierung interessanter paralleler und nebenläufiger Programme. Damit bestätigen wir also Avrons Vermutung.

In einem ersten Schritt gewinnen wir aus dem Hypersequentialkalkül für die betrachtete Logik einen geeigneten Kalkül des natürlichen Schließens. Um dies zu erreichen, übersetzen wir den Hypersequentialkalkül zunächst in einen Kalkül mit Regelsystemen. Hierbei handelt es sich um eine Erweiterung des Sequentialkalküls um Regeln, die globalen Nebenbedingungen unterliegen, ähnlich derjenigen wie sie in den Arbeiten Schroeder-Heisters zum natürlichen Schließen in höhereren Stufen vorkommen.

Wir beschreiben die Curry-Howard-Korrespondenz der erhaltenen Kalküle des natürlichen Schließens für die klassische Logik und die Logik von Gödel-Dummett. Beide Kalküle λ_{Cl} und λ_{G} erweitern den einfach typisierten λ-Kalkül durch einen Parallelitätsoperator, der die Kommunikation zwischen parallelen Prozessen ermöglicht. Wir beweisen für beide Systeme die Normalisierung, zeigen dass normalisierte Terme die Subformeleigenschaft besitzen, und erhalten damit die ersten Interpretation dieser bekannten Logiken als Systeme nebenläufiger Berechnung.

In der Folge verallgemeinern wir die obigen Ergebnisse, indem wir einen Erweiterung $\lambda_{\|}$ des λ-Kalküls durch Parallelismus beschrieben. Diese Erweiterung zeichnet sich durch unendlich viele Typisierungsregeln aus, welche solchen Axiomen entsprechen, die sich als Regel im Hypersequentialkalkül ausdrücken lassen. Die Regeln verallgemeinern jene von λ_{Cl} und λ_{G} und ermöglichen die Typisierung komplexer Kommunikationstopologien. Wir entwickeln einen Algorithmus, der aus *jeder* Kommunikationstopologie in Form eines direkten Graphen automatisch eine Typisierungsregel von $\lambda_{\|}$ berechnet. Umgekehrt können die durch die Regel typisierten Terme ausschließlich entsprechend der Topologie

kommunizieren. Die starke Normalisierung des Systems stellt sicher, dass alle Berechnungen – unabhängig von der angewandten Strategie – terminieren. Schließlich demonstrieren wir die Ausdrucksstärke von $\lambda_{\|}$ anhand mehrerer Beispiele paralleler Programme, die von numerischen Berechnungen bis zu Algorithmen auf Graphen reichen.

Der Kalkül $\lambda_{\|}$ nutzt Axiome intermediärer Logiken zur Typisierung paralleler Programme. Allerdings ermöglicht er keine computationale Interpretation der betrachteten Logiken, da seinem Typisierungssystem Regeln für die Disjunktion fehlen; selbige sind in vielen der Logiken nicht definierbar. Außerdem lässt sich aus der Normalisierung des Systems nicht die Subformeleigenschaft gewinnen. Wir überwinden diese Probleme durch die Beschreibung einer Curry-Howard-Korrespondenz für die vollständigen Logiken, und indem wir allgemeinere Reduktionen für die Kommunikationen zulassen. Diese Reduktionen, welche bereits in λ_{Cl} und λ_{G} vorkommen, gewährleisten die Subformeleigenschaft und implementieren Techniken der sogenannten Code Mobility für die Übertragung des Funktionsabschlusses.

Abstract

Avron speculated in 1991 that the proof-theoretical formalism of hypersequents might be deeply connected with concurrent computation. He suggested that a hypersequent can be seen as the parallel composition of several sequents and he envisaged the possibility of using the intermediate logics that can be naturally captured by hypersequent calculi "as bases for parallel λ-calculi" [Avr91].

In this dissertation we define parallel and concurrent λ-calculi based on Curry–Howard correspondences for intermediate logics that can be formalized as hypersequent calculi. The introduced calculi are more expressive than simply-typed λ-calculus and can formalize interesting parallel and concurrent programs. We thus confirm Avron's thesis.

Our first step in this direction is the extraction of suitable natural deduction calculi from hypersequent calculi for these logics. In order to do so, we first embed the hypersequent calculi into system of rules calculi: a proof-theoretical formalism which extends sequent calculus by rules featuring non-local conditions similar to those of higher-level natural deduction à la Schroeder-Heister.

We present Curry–Howard correspondences for the introduced natural deduction calculi for classical and Gödel–Dummett logic. The resulting calculi λ_{Cl} and λ_{G} are concurrent extensions of simply typed λ-calculus. We prove normalization results for both, show that normal proof terms enjoy the subformula property, and thus present the first concurrent computational interpretations of these well-known logics.

We then generalize the previous results by introducing a parallel extension of λ-calculus, denoted $\lambda_{\|}$, which features infinitely many type assignment rules corresponding to axioms that can be represented as hypersequent rules. These rules generalize those of λ_{Cl} and λ_{G}, and enable us to type complex communication topologies. We present an algorithmic procedure which extracts a $\lambda_{\|}$ type assignment rule from *any* communication topology that can be represented as a directed graph. The terms typed by the rule, in turn, can only communicate according to the topology. A strong normalization result for the system guarantees the termination of all computations, regardless of the employed strategy. The expressive power of $\lambda_{\|}$ is showcased by examples of parallel programs which range from numeric computation to algorithms on graphs.

The calculus $\lambda_{\|}$ successfully exploits axioms characterizing intermediate logics for typing parallel programs. Nevertheless, it does not provide a computational interpretation of the

considered logics because its type system does not contain disjunction rules, which are not definable in many of these logics, and its normalization does not imply the subformula property. We fill this gap by presenting Curry–Howard correspondences for the full logical systems and by introducing more general communication reductions. These reductions – also featured in λ_{Cl} and λ_{G} – ensure that the subformula property holds and implement code mobility techniques for function closure transmission.

Contents

CHAPTER 1

Introduction

Proofs have always been a central concern in mathematics, but only at the beginning of XX century they started to be studied as formal mathematical objects. This change of perspective is due to Hilbert, and gave rise to the meta-mathematical viewpoint which is the foundation of proof theory. Approximately half a century later the formal notion of proof used in this context has been associated with the notion of function and, not much later, of algorithm. The notion of *intuitionistic* proof was the first to be directly associated to that of computer program as formalized by λ-calculus. The development of a deep and fruitful research line followed the first results by Haskell Curry and William Howard [How80] associating these seemingly distant notions. Nowadays several different logics have been related to various notions of computation. These logics include classical logic [Gri90, Par92, dG95, Wad03, AZ16], linear logic [CP10a] and modal logics [MCHP04]; the list is long, and we refer the reader to [Wad15]. In-depth investigations of the computational aspects of intermediate logics, on the other hand, have been conducted only in the last few years [Asc16]. This class of logics contains all the propositional extensions of intuitionistic logic that are not stronger than classical logic and the first signs of interest in their computational content date back to 1991. In this year, Avron introduced the *hypersequent calculus* – a proof calculus based on multisets of sequents – for one of the best known intermediate logic: Gödel–Dummett logic, and speculated that there might be a strong connection between hypersequents and concurrent computation [Avr91]. He suggested that a hypersequent might be seen as the parallel composition of several sequents and he envisaged the possibility of using the intermediate logics that can be captured by hypersequent calculi "as bases for parallel λ-calculi".

In this dissertation we define parallel and concurrent λ-calculi based on Curry–Howard correspondences for intermediate logics that can be formalized as hypersequent calculi. We thus confirm Avron's thesis. The resulting calculi are strictly more expressive than

simply-typed λ-calculus and suitable systems for the representation of concurrent systems and for encoding parallel algorithms.

In order to define Curry–Howard correspondences for extracting the concurrent content of hypersequent proofs, we identify a class of proof systems which exhibit parallel features similar to those of hypersequent calculi but, at the same time, are simple enough to admit an elegant computational interpretation. We obtain this class by translating hypersequent calculi into suitable natural deduction calculi. To this aim, we first embed the hypersequent calculi into system of rules calculi [Neg16]: a proof-theoretical formalism which extends sequent calculus by rules featuring non-local conditions similar to those of higher-level natural deduction à la Schroeder-Heister [SH14]. Consider, for example, the hypersequent rule (1) – which is a version of the rule that Avron calls *communication* and characterizes Gödel–Dummett logic – the corresponding system of rules (2) and higher-level natural deduction rule (3):

$$(1)\quad \frac{G \mid B, \Sigma_1 \Rightarrow \Delta_1 \quad G \mid A, \Sigma_2 \Rightarrow \Delta_2}{G \mid A, \Sigma_1 \Rightarrow \Delta_1 \mid B, \Sigma_2 \Rightarrow \Delta_2}$$

$$(3)\quad \dfrac{\dfrac{\dfrac{A}{B}\,\dagger \qquad \dfrac{B}{A}\,\dagger}{\begin{array}{cc} \vdots & \vdots \\ C & C \end{array}}\,\star}{C}$$

$$(2)\quad \dfrac{\dfrac{\dfrac{B, \Sigma_1 \Rightarrow \Delta_1}{A, \Sigma_1 \Rightarrow \Delta_1}\,\dagger \qquad \dfrac{A, \Sigma_2 \Rightarrow \Delta_2}{B, \Sigma_2 \Rightarrow \Delta_2}\,\dagger}{\begin{array}{cc} \vdots & \vdots \\ \Gamma \Rightarrow \Pi & \Gamma \Rightarrow \Pi \end{array}}\,\star}{\Gamma \Rightarrow \Pi}$$

where the schemata (2) and (3) mean that the rules marked by † can only be applied in the subtree above the relative premise of $\star$. As we will formally show in Chapter 3, the structural connective $\mid$ of hypersequents – interpreted as a disjunction – simply allows us to represent in a local way the connection between the sequent rules marked by † in (2). The global structure of the system of rules reflects, in turn, the structure of the higher-level natural deduction rules. Following these intuitions, we show the equivalence between the hypersequent formalism and a fragment of the system of rules formalism, and provide a translation of hypersequent rules into higher-level natural deduction rules.

Using the obtained natural deduction calculi for classical and Gödel–Dummett logic, we then define Curry–Howard correspondences resulting in two typed concurrent extensions of λ-calculus: λ_{Cl} and λ_{G}, respectively. Although the computational content of classical proofs has been investigated for a long time, the computational interpretation of classical logic that we present here is the first in terms of concurrent computation. On the other hand, no Curry–Howard correspondence had been defined for Gödel–Dummett logic despite the attempts in [BCF00], [Hir12] and [BP15].

From a computational viewpoint, both λ_{Cl} and λ_{G} can encode parallel processes connected by higher-order communication channels for the trasmission of terms. While λ_{Cl} only features unidirectional communication channels connecting two processes, as represented below on the left, λ_{G} features symmetric channels, as shown below on the right:

From a technical point of view, in λ_{Cl} we can type a term of the form $\mathcal{C}[\overline{a}\ t] \parallel_a \mathcal{D}$ such that a occurs in $\mathcal{D}$. This term represents the parallel composition of the processes $\mathcal{C}[\overline{a}\ t]$ and $\mathcal{D}$ and their connection by a communication channel a, which behaves similarly to a π-calculus binder (νa). The λ_{Cl} reduction

$$\mathcal{C}[\overline{a}\ t] \parallel_a \mathcal{D} \ \mapsto_{\mathtt{c}}\ \mathcal{D}[t/a]$$

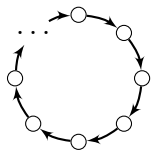
formalizes then the transmission of the term t from the process $\mathcal{C}[\overline{a}\ t]$ to the process $\mathcal{D}$. For a λ_{G} term of the form $\mathcal{C}[a\,u] \parallel_a \mathcal{D}[a\,v]$, on the other hand, we have both reductions for transmitting the message u from left to right

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

and reductions for transmitting the message v from right to left

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

Generalizing the ideas at the base of λ_{Cl} and λ_{G} but strengthening the features that are relevant for parallel programming, we obtain the typed parallel λ-calculus $\lambda_{\parallel}$. On the one hand, the type system of $\lambda_{\parallel}$ presents strong simplifications with respect to those of λ_{Cl} and λ_{G}, on the other hand, its individual type assignment rules are based on a rather general class of axioms that can be formalized as hypersequent rules and enable us to type whole network topologies. For instance, a single application of a $\lambda_{\parallel}$ type assignment rule enables us to type networks based on the following graph:



In general, we can automatically extract a $\lambda_{\parallel}$ type assignment rule from *any* communication topology that can be represented as a directed graph. The possibility of typing at once a whole network greatly increases the control over the interactions between processes, and thus makes $\lambda_{\parallel}$ ideal for encoding parallel algorithms. The restrictions on $\lambda_{\parallel}$ type system, moreover, enable us to show a strong normalization result which guarantees that all possible reduction strategies for $\lambda_{\parallel}$ terms are terminating. To show the expressivity of $\lambda_{\parallel}$, we encode several algorithms, including a parallel version of Floyd–Warshall algortihm, see [Loo11], for computing the shortest path between two nodes in a graph.

The strong computational properties of $\lambda_{\parallel}$, though, come at the cost of its unsuitability as a computational interpretation for the considered intermediate logics. The type system of $\lambda_{\parallel}$ does not contain disjunction rules, which are not definable in many of these logics.

Moreover, the normalization of $\lambda_{\parallel}$ does not imply the subformula property. We fill this gap by presenting $\lambda_{\parallel \mathrm{L}}$, a framework for Curry–Howard correspondences for intermediate logics, and by introducing more general communication reductions. These reductions – also featured in λ_{Cl} and λ_{G} – ensure that the subformula property holds and enable the communication of terms that depend on their computational environment. The problem of restoring the required dependencies after such a communication is often called the problem of the *transmission of closures*. This is, as explained in [EBPJ11], a "*fundamental problem in any distributed implementation of a statically-typed, higher-order programming language*". Our proof systems provide a solution to this problem: a new communication channel is established on the fly in order to handle the dependencies, or *closure*, of the transmitted functions. Consider for example the λ_{Cl}-term

$$\mathcal{C}[\lambda y\, a(y+2)] \parallel_a \mathcal{D}[a+3]$$

in which the two parallel processes $\mathcal{C}[\lambda y\, a(y+2)]$ and $\mathcal{D}[a+3]$ are connected by the communication channel a. Suppose for the sake of simplicity that only one instance of the variable a occurs in each of them and that natural numbers and addition are defined in our calculus. If we simply transmit the message $y+2$ to the other term and obtain $\mathcal{D}[(y+2)+3]$, we violate the dependence of y on λy. To avoid this, the following reduction is triggered

$$\mathcal{C}[\lambda y\, a(y+2)] \parallel_a \mathcal{D}[a+3] \qquad \mapsto_{\mathsf{c}}^{*} \qquad \mathcal{C}[\lambda y\, by] \parallel_b \mathcal{D}[(b+2)+3]$$

in which the message $y+2$ is transmitted to the other process, but the new channel b is established to keep track of the dependence of $y+2$ – which is now $b+2$ – on $\lambda y\, by$. Thus, when the value of y will be available in $\mathcal{C}[\lambda y\, by]$, it will be transmitted through b to $\mathcal{D}[(b+2)+3]$. For instance, if

$$\mathcal{C}[\lambda y\, by] \parallel_b \mathcal{D}[(b+2)+3] \qquad \mapsto_{\mathsf{c}}^{*} \qquad \mathcal{C}'[b1] \parallel_b \mathcal{D}[b+5]$$

then we have the λ_{Cl} communication

$$\mathcal{C}'[b1] \parallel_b \mathcal{D}[b+5] \qquad \mapsto_{\mathsf{c}}^{*} \qquad \mathcal{D}[1+5]$$

Finally, while the reduction conditions of the calculi λ_{Cl}, λ_{G} and $\lambda_{\parallel}$ are essentially based on the type of terms, $\lambda_{\parallel \mathrm{L}}$ reduction rules are purely based on the syntactic structure of terms and make no reference to their type. Therefore the set of reduction rules of $\lambda_{\parallel \mathrm{L}}$ shows a possible approach to the problem of defining a well-behaved untyped concurrent λ-calculus.

1.1 Outline of the dissertation

The structure of the dissertation is the following.

In Chapter 2 we define logics, proof-theoretical calculi and formalisms used throughout this work, we present the reasons that led to their introduction and the background of the problems addressed in the dissertation.

In Chapter 3 we present a double embedding between hypersequent calculi and sequent calculi with two-level system of rules, which is a proper fragment of the formalism of system of rules. This result consists in a translation that produces from any calculus in one of the two formalisms an equivalent calculus in the other. By exploiting the similarity between the non-local conditions of systems of rules and natural deduction discharge mechanism, we introduce a translation from hypersequent rules to higher-level natural deduction rules.

Chapter 4 is devoted to the typed concurrent λ-calculi λ_{Cl} (Section 4.1) and λ_{G} (Section 4.2). They are introduced by defining Curry–Howard correspondences on natural deduction calculi for classical logic and Gödel–Dummett logic, respectively. We introduce the reduction systems for the calculi and we discuss the general communication reductions – called *cross reductions* – that enable us to transmit terms that depend on their computational environment. We prove that the subformula property holds for normal proof terms. The normalization proof is based on the same technique for both calculi and uses the subformula property as a computational criterion for triggering communication reductions. We analyze the expressive power of the calculi and present some examples of computations. A comparison of λ_{Cl}, λ_{G} and existing related calculi concludes the chapter.

In Chapter 5 we introduce the parallel λ-calculus $\lambda_{\parallel}$. We define the calculus by computationally interpreting an infinite class of natural deduction rules corresponding to axioms that can be formalized as hypersequent rules. Exploiting the generality of the class of axioms on which $\lambda_{\parallel}$ is based, we provide an algorithm for extracting $\lambda_{\parallel}$ type assignment rules from network topologies that can be represented as directed graphs. Moreover, we show that the terms typed by these rules will only communicate according to the topologies. We then prove the strong normalization result for $\lambda_{\parallel}$ by using the Tait-Girard reducibility technique of [GLT89]. We use $\lambda_{\parallel}$ to encode parallel algorithms ranging from numeric computation to graph analysis. Finally, we compare $\lambda_{\parallel}$ to similar existing systems.

In Chapter 6 we present the general framework $\lambda_{\parallel\mathrm{L}}$ for defining Curry–Howard correspondences for intermediate logics that can be formalized by hypersequent calculi. We present the type system of $\lambda_{\parallel\mathrm{L}}$ which, unlike the type system of $\lambda_{\parallel}$, also includes disjunction rules, and we discuss its correspondence with complete natural deduction systems for the considered logics. We explain the differences between the reduction rules of $\lambda_{\parallel\mathrm{L}}$, which are exclusively based on the shape of terms and do not make any reference to their type, and those of λ_{Cl}, λ_{G} and $\lambda_{\parallel}$. Finally, we prove that the reduction system of $\lambda_{\parallel\mathrm{L}}$ is terminating and that normal $\lambda_{\parallel\mathrm{L}}$ proof terms enjoy the subformula property.

We conclude and discuss several future research directions in Chapter 7.

1.2 Publications

This dissertation is based on the following publications.

1. Gödel logic: from natural deduction to parallel computation. (Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco.) In *LICS 2017*, pages 1–12, 2017.

2. Classical proofs as parallel programs. (Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco.) In *GandALF 2018*, pages 43–57, 2018.

3. From intermediate logics to parallel λ-calculi via Curry–Howard correspondences. (Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco.) Unpublished, 2018.

4. A parallel λ-calculus for graph-based communication. (Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco.) Submitted for publication, 2018.

5. Embedding formalisms: hypersequents and two-level systems of rules. (Agata Ciabattoni and Francesco A. Genco.) In *Advances in Modal Logic*, volume 11, pages 197–216. College Publications, 2016.

6. Hypersequents and systems of rules: Embeddings and applications. (Agata Ciabattoni and Francesco A. Genco.) *ACM Transactions on Computational Logic (TOCL)*, 19(2):11:1–11:27, 2018.

CHAPTER 2

Logics, calculi and computations

We present now the logics and calculi to which we refer throughout the dissertation, we establish the relative notation, and we briefly mention the historical reasons that led to the introduction of these systems. Moreover, we present and discuss the background of the problems addressed by the dissertation.

In particular, in Section 2.1 we present intuitionistic logic, classical logic and some intermediate logics which are relevant for the present work. We define here the language of these logics and provide some insights in their semantics. In Section 2.2 we present sequent calculus and the generalization of this calculus to multisets of sequents: hypersequent calculus. In Section 2.3 we present and discuss sequent calculi with systems of rules; another, more recent, generalization of sequent calculi. Section 2.4 is devoted to natural deduction calculi and to the Curry–Howard, or proofs-as-programs, correspondence. We first present natural deduction calculus and then, in Section 2.4.1, simply typed λ-calculus and the Curry–Howard correspondence for intuitionistic logic. In Section 2.4.2 we discuss some prominent Curry–Howard correspondences for classical logic. Finally, in Section 2.4.3 we discuss Avron's thesis on the possibility of providing concurrent computational interpretations of the intermediate logics that can be formalized as hypersequent calculi. In doing this, we also discuss some related attempts to define suitable natural deduction calculi for intermediate logics.

2.1 Intuitionistic and intermediate logics

The alphabet of propositional intuitionistic logic (**IL**) consist of an enumerable set of propositional variables $\{p_1, p_2, \ldots\}$ and the logical connectives $\rightarrow, \wedge, \vee, \bot$. The language of **IL** is inductively defined as follows:

- $p_i \in \{p_1, p_2, \ldots\}$ and $\bot$ are formulae;

- if A and B are formulae, then $A \to B$, $A \wedge B$, $A \vee B$ are formulae as well.

Notation. We assume that the connectives $\to$ and $\wedge$ associate to the right. Moreover, as usual, we define $\neg A$ as $A \to \bot$, and the true proposition $\top$ as $\bot \to \bot$ and we adopt the convention that the empty conjunction, namely the expression $A_1 \wedge \ldots \wedge A_n$ for $n = 0$, denotes $\top$.

Unless stated otherwise, we generally use the upper-case Latin letters A, B, C, D, E and F for formulae, upper-case Greek letters for multisets of formulae, and, in particular, we use Π for those multisets of formulae that contain at most one element.

Intuitionistic logic

The first formal definition of predicative intuitionistic logic [Hey30] is in the form of an axiomatic system, or Hilbert system. An axiomatization of intuitionistic logic in this style contains the rule

$$\frac{A \quad A \to B}{B}\ (\mathrm{MP})$$

and the following axiom schemata:

$$\bot \to A \qquad A \to (B \to A) \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C))$$
$$A \wedge B \to A \qquad A \wedge B \to B \qquad A \to (B \to A \wedge B)$$
$$A \to A \vee B \qquad B \to A \vee B \qquad (A \to C) \to ((B \to C) \to (A \vee B \to C))$$

A proof in this system is a list of formulae which are either (i) instances of an axiom, or (ii) obtained applying (MP) to elements occurring earlier in the list. To prove a formula A, we need to construct a proof that contains A.

Notation. We often use the notion of *instance* of an axiom or rule schema. Such an instance is obtained, as usual, by replacing uniformly all metavariables for formulae with formulae of the language.

The logic **IL** can then be defined as the set of all formulae that are provable in this system. The conceptual meaning of intuitionistic connectives, though, is best captured by an informal interpretation provided in 1934 by Arend Heyting and later known as Brouwer-Heyting-Kolmogorov (BHK) interpretation. One version of the BHK interpretation for propositional connectives is the following:

- there is no proof of $\bot$
- a proof of $A \wedge B$ is a pair containing a proof of A and a proof of B
- a proof of $A \vee B$ is either a proof of A or a proof of B
- a proof of $A \to B$ is a construction that transforms any proof of A into a proof of B

The BHK interpretation provides an informal description of the meaning of the connectives of intuitionistic logic in terms of proofs and constructions. Just looking at this

interpretation we can quite distinctly perceive its procedural flavor, and we will see in Section 2.4 that BHK can indeed be used as a series of guidelines for constructing proofs and for interpreting such proofs as algorithms. In Section 2.4 we will present a proof system closer to this interpretations of intuitionistic connectives.

Intuitionistic logic can also be characterized by a *frame semantics*, or *relational semantics*; see for instance Chapter 2 of [CZ97]. This semantics enables us to define relational models consisting of a *frame* (W, R) – where W is a set and R is a partial order over W – and a valuation function which associates each propositional variable with a subset of W. Conditions for the connectives are provided in such a way that we can lift each valuation to compound formulae and thus check weather a formula is true in a model. We can then define **IL** as the set of formulae which are true in all models.

Classical logic

If we add the axiom $\text{EM} = A \vee \neg A$ to the axiomatic system for intuitionistic logic we obtain an axiomatization for classical logic (**CL**). This axiom is known as the law of the excluded middle and, intuitively, narrows down the possibilities of semantical interpretation of each formula to two cases: the formula is either true or false. This intuition is clearly reflected in the semantics of classical logic: a classical valuation v is a function that maps formulae into the set $\{0, 1\}$ of truth values in such a way that

$$\begin{array}{lcl} v(A \vee B) & = & max(v(A), v(B)) \\ v(A \wedge B) & = & min(v(A), v(B)) \\ v(A \to B) & = & \begin{cases} 0 \text{ if } v(A) = 1 \text{ and } v(B) = 0 \\ 1 \text{ otherwise} \end{cases} \\ v(\neg A) & = & \begin{cases} 0 \text{ if } v(A) = 1 \\ 1 \text{ if } v(A) = 0 \end{cases} \end{array}$$

If a formula is true – namely, has value 1 – with respect to all valuations, then it is a theorem of classical logic.

Gödel–Dummet logic and other intermediate logics

An intermediate logic is any consistent logic whose theorems are at least all theorems of intuitionistic logic and at most all theorems of classical logic. Intermediate logics, as classical logic, will be formally defined by extending proof systems for intuitionistic logic.

Both intuitionistic logic and classical logic, technically, are intermediate logics, but a more interesting exemplar belonging to this class – and a central one for this dissertation – is Gödel–Dummett logic (**GL**). This logic is going to be analyzed from a computational perspective in Chapter 4 and is among the simplest and more studied intermediate logics. The semantics originally defining it was introduced by Gödel in [Göd32] to prove that the connectives of intuitionistic logic cannot be described by finite matrices. To this aim, Gödel defined a sequence of increasingly strong intermediate logics, nowadays called

k-valued Gödel logics ($\mathbf{G}_k$). Fixed a natural number k, the k-valued Gödel logic is semantically characterised by the matrix

$$\begin{array}{lcl} v(A \vee B) & = & min(v(A), v(B)) \\ v(A \wedge B) & = & max(v(A), v(B)) \\ v(A \to B) & = & \begin{cases} 0 \text{ if } v(A) \geq v(B) \\ v(B) \text{ if } v(A) < v(B) \end{cases} \\ v(\neg A) & = & \begin{cases} k \text{ if } v(A) < k \\ 0 \text{ if } v(A) = k \end{cases} \end{array}$$

for functions v mapping formulae into the set $\{1, \ldots, k\}$ of truth values. If a formula has value 1 for all valuations, then it is a theorem of the k-valued Gödel logic.

Later Dummett generalized Gödel's idea to infinite matrices taking $k = \omega$, and officially introduced Gödel–Dummett logic under the name of **LC** [Dum59].

Axiomatically, Gödel–Dummett logic can be characterized as an extension of intuitionistic logic by the linearity axiom $\text{Lin} = (A \to B) \vee (B \to A)$. For the present work, we will adopt this definition.

Other intermediate logics can be defined in reference to the frame semantics of intuitionistic logic, as for example $\mathbf{BC}_k$ and $\mathbf{BW}_k$. The former is the logic of intuitionistic frames with cardinality at most k. In other terms the theorems of $\mathbf{BC}_k$ are the formulae which are valid in all relational models based on a frame with at most k nodes. The logic $\mathbf{BW}_k$, on the other hand, contains the formulae which are valid on all intuitionistic frames with width at most k; where the width of a frame is the cardinality of a maximal set of pairwise unrelated nodes.

2.2 Sequents and hypersequents

Sequents have been introduced in the context of Gentzen's proof of the consistency of arithmetic as a means to understand purely logical deductions. A *sequent* [Gen35] is an object of the form

$$\Gamma \Rightarrow \Pi$$

where Γ is a possibly empty multiset of formulae in the language of intuitionistic logic. We adopt the restriction here that Π contains at most one formula.

In the context of intuitionistic logic, a sequent $\Gamma \Rightarrow \Pi$ is interpreted as the formula $\bigwedge \Gamma \to A$ where $\bigwedge \Gamma$ is the conjunction of elements of Γ and A is either the unique element of Π or $\bot$ if Π contains no elements.

A proof calculus is a set of rules for constructing derivations: formal objects witnessing the fact that a formula is a theorem of a logic. We define now the notion of local proof calculus, which will be enough to introduce both sequent calculi and hypersequent calculi.

Definition 2.2.1 (Local proof calculus)**.** For any class O of objects, a *local proof calculus* $\mathcal{C}$ for constructing derivations of objects O is a set of rules of the form $\frac{o_1 \dots o_n}{o}$ where $o, o_1 \dots o_n \in O$. A derivation of $o \in O$ is then inductively defined as follows:

- if, for some $n \geq 0$
$$\frac{o_1 \dots o_n}{o} \in \mathcal{C}$$
and $\mathcal{D}_1, \dots, \mathcal{D}_n$ are derivations of $o_1, \dots, o_n$ respectively, then
$$\frac{\mathcal{D}_1 \dots \mathcal{D}_n}{o}$$
is a derivation of o.

The rules of the sequent calculus L$\mathfrak{I}$ [Gen35] for intuitionistic logic are shown in Table 2.1. Rules are presented, as usual, as rule schemata and we omit the inference line of rules without premises. Following standard practice, when unambiguous, we do not explicitly distinguish between a rule and a rule schema.

$$A \Rightarrow A \quad \bot \Rightarrow \Pi \qquad \frac{\Gamma, A \Rightarrow \Pi \quad \Gamma, B \Rightarrow \Pi}{\Gamma, A \vee B \Rightarrow \Pi}\ (\vee l) \qquad \left\{\frac{\Gamma \Rightarrow A_i}{\Gamma \Rightarrow A_1 \vee A_2}\ (\vee r)\right\}_{i \in \{1,2\}} \qquad \frac{\Gamma \Rightarrow \Pi}{A, \Gamma \Rightarrow \Pi}\ (\mathrm{IW})$$

$$\frac{\Gamma, A, B \Rightarrow \Pi}{\Gamma, A \wedge B \Rightarrow \Pi}\ (\wedge l) \qquad \frac{\Gamma \Rightarrow A \quad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B}\ (\wedge r) \qquad \frac{A, A, \Gamma \Rightarrow \Pi}{A, \Gamma \Rightarrow \Pi}\ (\mathrm{IC})$$

$$\frac{\Gamma \Rightarrow A \quad \Gamma, B \Rightarrow \Pi}{\Gamma, A \to B \Rightarrow \Pi}\ (\to l) \qquad \frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \to B}\ (\to r) \qquad \frac{\Gamma \Rightarrow A \quad A, \Delta \Rightarrow \Pi}{\Gamma, \Delta \Rightarrow \Pi}\ (\mathrm{cut})$$

Table 2.1: The sequent calculus L$\mathfrak{I}$.

A formula A is a theorem of intuitionistic logic if and only if there exists a derivation of $\Rightarrow A$ in L$\mathfrak{I}$, see for example [TS96].

The main technical result that Gentzen showed about L$\mathfrak{I}$ – which is also the main technical tool that he used to prove the consistency of arithmetic – is the *cut-elimination* theorem. This theorem states that we can constructively remove all (cut) rule applications from any proof in the calculus. For the systems considered here, this implies the *subformula property*: every derivable formula can be proved by a derivation that only contains subformulae of the formula itself. A proof that only contains subformulae of its conclusion is also called an *analytic* proof, in the sense that the proof proceeds by analysis of the statement to be proved and does not involve external concepts which are not already mentioned in the statement itself.

The field of application of sequent calculi is certainly not restricted to the study of intuitionistic logic: they are versatile objects which have been employed to define calculi for several logics. Nonetheless, they lack the expressive power required by certain tasks. For instance, if we want to define modular analytic calculi, for some classes of logics,

sequents are not expressive enough. For this reason various generalization of sequent calculi have been introduced and hypersequents are one of these. Hypersequents are a rather simple generalization of sequents which have proved very well suited for capturing a large class of intermediate logics.

A *hypersequent* [Avr87, Avr91] is a multiset of sequents. Hypersequents are usually represented as objects of the form

$$\Gamma_1 \Rightarrow \Pi_1 \mid \ldots \mid \Gamma_n \Rightarrow \Pi_n$$

where $\Gamma_1 \Rightarrow \Pi_1, \ldots, \Gamma_n \Rightarrow \Pi_n$ are sequents and are called the *components* of the hypersequent.

The interpretation of a hypersequent $\Gamma_1 \Rightarrow \Pi_1 \mid \ldots \mid \Gamma_n \Rightarrow \Pi_n$ will be the formula $(\bigwedge \Gamma_1 \Rightarrow A_1) \vee \ldots \vee (\bigwedge \Gamma_n \Rightarrow A_n)$ where $\bigwedge \Gamma_i$ is the conjunction of elements of Γ_i and A_i is either the unique element of Π_i or $\bot$ if Π_i contains no elements.

Notation. We generally use the letters G and H to denote possibly empty hypersequents.

The rules of the hypersequent calculus H$\mathfrak{I}$ for intuitionistic logic are shown in Table 2.2.

$$A \Rightarrow A \quad \bot \Rightarrow \Pi \quad \frac{G \mid \Gamma, A \Rightarrow \Pi \quad G \mid \Gamma, B \Rightarrow \Pi}{G \mid \Gamma, A \vee B \Rightarrow \Pi}\ (\vee l) \quad \left\{\frac{G \mid \Gamma \Rightarrow A_i}{G \mid \Gamma \Rightarrow A_1 \vee A_2}\ (\vee r)\right\}_{i \in \{1,2\}}$$

$$\frac{G \mid \Gamma, A, B \Rightarrow \Pi}{G \mid \Gamma, A \wedge B \Rightarrow \Pi}\ (\wedge l) \quad \frac{G \mid \Gamma \Rightarrow A \quad G \mid \Gamma \Rightarrow B}{G \mid \Gamma \Rightarrow A \wedge B}\ (\wedge r) \quad \frac{G \mid \Gamma \Rightarrow \Pi}{G \mid A, \Gamma \Rightarrow \Pi}\ (\mathrm{IW})$$

$$\frac{G \mid \Gamma \Rightarrow A \quad G \mid \Gamma, B \Rightarrow \Pi}{G \mid \Gamma, A \to B \Rightarrow \Pi}\ (\to l) \quad \frac{G \mid \Gamma, A \Rightarrow B}{G \mid \Gamma \Rightarrow A \to B}\ (\to r) \quad \frac{G \mid A, A, \Gamma \Rightarrow \Pi}{G \mid A, \Gamma \Rightarrow \Pi}\ (\mathrm{IC})$$

$$\frac{G \mid \Gamma \Rightarrow A \quad G \mid A, \Delta \Rightarrow \Pi}{G \mid \Gamma, \Delta \Rightarrow \Pi}\ (\mathrm{cut}) \quad \frac{G}{G \mid \Gamma \Rightarrow \Pi}\ (\mathrm{EW}) \quad \frac{G \mid \Gamma \Rightarrow \Pi \mid \Gamma \Rightarrow \Pi}{G \mid \Gamma \Rightarrow \Pi}\ (\mathrm{EC})$$

Table 2.2: The hypersequent calculus H$\mathfrak{I}$.

Sequent and hypersequent rules can be divided into *logical rules* and *structural rules*. If a logical connective is shown in the schema of a rule, then we call such rule *logical*; otherwise, we call the rule *structural*.

Note that the added structure of H$\mathfrak{I}$ with respect to the sequent calculus L$\mathfrak{I}$ [Gen35] for intuitionistic logic is in fact redundant since a hypersequent $\Gamma_1 \Rightarrow \Pi_1 \mid \ldots \mid \Gamma_k \Rightarrow \Pi_k$ is derivable in H$\mathfrak{I}$ if and only if $\Gamma_i \Rightarrow \Pi_i$ is derivable in L$\mathfrak{I}$ for some $i \in \{1, \ldots, k\}$. Indeed, any sequent calculus can be trivially viewed as a hypersequent calculus. The added expressive power of the latter is due to the possibility of defining new rules which act simultaneously on several components of one or more hypersequents.

Example 2.2.1. By adding to H$\mathfrak{J}$ the following version of the structural rule introduced in [Avr91]

$$\frac{G \mid \Phi, \Gamma_1 \Rightarrow \Pi_1 \quad G \mid \Psi, \Gamma_2 \Rightarrow \Pi_2}{G \mid \Psi, \Gamma_1 \Rightarrow \Pi_1 \mid \Phi, \Gamma_2 \Rightarrow \Pi_2}\ (\text{com})$$

we obtain a cut-free calculus for Gödel–Dummett logic (**GL**), which can be defined extending intuitionistic logic by the linearity axiom Lin: $(A \to B) \vee (B \to A)$. For example, a derivation of Lin using (com) is the following

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B \qquad A \Rightarrow A}{A \Rightarrow B \mid B \Rightarrow A}\ (\text{com})}{A \Rightarrow B \mid\Rightarrow B \to A}}{\Rightarrow A \to B \mid\Rightarrow B \to A}}{\Rightarrow A \to B \mid\Rightarrow (A \to B) \vee (B \to A)}}{\Rightarrow (A \to B) \vee (B \to A) \mid\Rightarrow (A \to B) \vee (B \to A)}}{\Rightarrow (A \to B) \vee (B \to A)}$$

As the usual interpretation of the symbol | is disjunctive, the hypersequent calculus can naturally capture properties that can be expressed in a disjunctive form, such as certain Hilbert axioms and algebraic equations, see [CGT08].

Definition 2.2.2. Given a hypersequent rule (r) with premises $G \mid H_1 \ \ldots \ G \mid H_n$ and conclusion $G \mid H$, we call *active components* the components of the hypersequents $H_1, \ldots, H_n, H$ and *context components* the components of G.

Notice that each hypersequent rule considered in this section and in Chapter 3 have (i) a shared external context, namely the same context G for all premises, and (ii), except for (EC), at most one active component for each premise. Note that (i) is not a restriction and, in absence of eigenvariable conditions, neither is (ii) because in a hypersequent calculus containing (EC) and (EW), any rule is equivalent to one that satisfies these conditions.

Notation. Given a sequent or hypersequent calculus $\mathcal{C}$, we denote by $\vdash_{\mathcal{C}} o$ that there is a derivation of o in $\mathcal{C}$ and by $\vdash_{\mathcal{C}}^{c.f.} o$ that there is a derivation of o in $\mathcal{C}$ that does not contain applications of (cut). Moreover, given a set $\mathbb{R}$ of rules we denote by $\mathcal{C} + \mathbb{R}$ the calculus obtained by adding the elements of $\mathbb{R}$ to $\mathcal{C}$.

As usual, by the *height* of a sequent or hypersequent derivation we mean the maximal number of rule applications occurring on any of its branches plus 1.

A method is introduced in [CGT08] to transform propositional Hilbert axioms in the language of full Lambek calculus into equivalent hypersequent rules.

The method is based on the following classification of intuitionistic formulae: $\mathcal{N}_0$ and $\mathcal{P}_0$ are the set of atomic formulae and

axiom	*hypersequent rule*	*logic*
$A \vee \neg A$	$\dfrac{G \mid \Gamma, \Sigma \Rightarrow \Delta}{G \mid \Gamma \Rightarrow \mid \Sigma \Rightarrow \Delta}$	**CL**
$(A \to B) \vee (B \to A)$	$\dfrac{G \mid \Gamma, B \Rightarrow \Delta \quad G \mid \Sigma, A \Rightarrow \Theta}{G \mid \Gamma, A \Rightarrow \Delta \mid \Sigma, B \Rightarrow \Theta}$	**GL**
$\bigvee_{i=0}^{k-1}(A_i \to A_{i+1}) \vee (\neg A_k)$	$\dfrac{\{G \mid \Gamma_{j+1}, \Delta_j \Rightarrow \Pi_j\}_{j\in\{0,\dots,k-1\}}}{G \mid \Gamma_0, \Delta_0 \Rightarrow \Pi_0 \mid \dots \mid \Gamma_{k-1}, \Delta_{k-1} \Rightarrow \Pi_{k-1} \mid \Gamma_k \Rightarrow}$	$\mathbf{G}_k$
$\bigvee_{i=1}^{k-1}(A_i \to A_{i+1}) \vee (A_k \to A_1)$	$\dfrac{\{G \mid \Gamma_{j+1}, \Delta_j \Rightarrow \Pi_j\}_{j\in\{1,\dots,k-1\}} \quad G \mid \Gamma_1, \Delta_k \Rightarrow \Pi_k}{G \mid \Gamma_1, \Delta_1 \Rightarrow \Pi_1 \mid \dots \mid \Gamma_k, \Delta_k \Rightarrow \Pi_k}$	$\mathbf{C}_k$ [1]
$\bigvee_{i=0}^{k}(A_i \to \bigvee_{i\neq j=0}^{k} A_j)$	$\dfrac{\{G \mid \Gamma_i, \Gamma_j \Rightarrow \Delta_i\}_{i,j\in\{0,\dots,k\},\, i\neq j}}{G \mid \Gamma_0 \Rightarrow \Delta_0 \mid \dots \mid \Gamma_k \Rightarrow \Delta_k}$	$\mathbf{BW}_k$
$A_0 \vee \bigvee_{i=1}^{k}(\bigwedge_{j=0}^{i-1} A_j \to A_i)$	$\dfrac{\{G \mid \Gamma_i, \Gamma_j \Rightarrow \Delta_i\}_{i\in\{0,\dots,k-1\},\, i<j\leq k}}{G \mid \Gamma_0 \Rightarrow \Delta_0 \mid \dots \mid \Gamma_{k-1} \Rightarrow \Delta_{k-1} \mid \Gamma_k \Rightarrow}$	$\mathbf{BC}_k$

Table 2.3: Hypersequent rules and corresponding axioms.

$$\begin{array}{lll} \mathcal{P}_{n+1} & ::= \bot \mid \top \mid \mathcal{N}_n \mid \mathcal{P}_{n+1} \wedge \mathcal{P}_{n+1} \mid \mathcal{P}_{n+1} \vee \mathcal{P}_{n+1} \\ \mathcal{N}_{n+1} & ::= \bot \mid \top \mid \mathcal{P}_n \mid \mathcal{N}_{n+1} \wedge \mathcal{N}_{n+1} \mid \mathcal{P}_{n+1} \to \mathcal{N}_{n+1} \end{array}$$

We recall that a connective is said to be *positive* when its left logical rule is invertible and *negative* when its right logical rule is invertible [And92]. Thus, the classes $\mathcal{P}_n$ and $\mathcal{N}_n$ contain axioms with outermost connective positive and negative, respectively, since in H$\mathfrak{I}$ $(\vee l), (\to r), (\wedge l)$ and $(\wedge r)$ are invertible rules. Notice that $\wedge$ is both positive and negative because both rules for this connective are invertible in H$\mathfrak{I}$.

As shown in [CGT08] all axioms in the class $\mathcal{P}_3$ can be algorithmically transformed into equivalent *structural* hypersequent rules that are analytic, namely that preserve cut-elimination when added to the calculus H$\mathfrak{I}$. For instance the rule (com) in Example 2.2.1 can be automatically extracted from the linearity axiom Lin.

2.3 Systems of rules

Systems of rules were introduced in [Neg16] to define analytic labelled sequent calculi for logics that can be characterized imposing certain restrictions to the frame semantics [CZ97]

[1] See [LE82a].

for the modal logic **K**. In general, labelled calculi allow us to represent features of the relational models inside the machinery of the sequent calculus and thus enable us to define proof calculi based on certain frame conditions. Using systems of labelled rules we enlarge the class of frame conditions, and hence of logics, that we can capture by a labelled calculus. In particular, the frame conditions that we can capture by systems of labelled sequent rules are those that can be described by generalized geometric implications, a class of first-order formulae that goes beyond the fragment of geometric implications [Neg05] and that includes all frame properties that correspond to formulae in the Sahlqvist fragment. The class GA_0 of geometric implications is defined as containing formulae of the following form:

$$\mathrm{GA}_0 \;\equiv\; \forall \overline{x}(\bigwedge P \to \exists \overline{y}_1 \bigwedge P_1 \vee \ldots \vee \exists \overline{y}_m \bigwedge P_m)$$

where $\overline{x}, \overline{y}_1, \ldots, \overline{y}_m$ are tuples of first order variables, $\bigwedge P$ is a finite conjunction of atomic formulae, the variables in $\overline{y}_i$ for any i do not occur free in $\bigwedge P$, and $\bigwedge P_j$ is a conjunction of atomic formulae for any j. The first level of the hierarchy of generalized geometric implications is the class GA_1 and it is defined as containing formulae of the following form:

$$\mathrm{GA}_1 \;\equiv\; \forall \overline{x}(\bigwedge P \to \exists \overline{y}_1 \bigwedge \mathrm{GA}_0 \vee \ldots \vee \exists \overline{y}_m \bigwedge \mathrm{GA}_0)$$

where $\overline{x}, \overline{y}_1, \ldots, \overline{y}_m$ are tuples of first order variables, $\bigwedge P$ is a finite conjunction of atomic formulae, the variables in $\overline{y}_i$ for any i do not occur free in $\bigwedge P$, and $\bigwedge \mathrm{GA}_0$ is a finite conjunction of geometric implications. The nth level of the hierarchy is defined replacing the conjunctions of formulae GA_0 in the previous definition by conjunctions of formulae GA_i for $i < n$.

From a proof-theoretical viewpoint, the system of rules formalism combines the bookkeeping machinery of sequent calculi with a generalized version of the discharging mechanism of natural deduction. More precisely, a system of rules is a set of rules that can only be applied in a certain order and possibly share metavariables for formulae or sets of formulae. Consider for example the following system of sequent rules:

$$\dfrac{\dfrac{\Sigma, \Gamma_1 \Rightarrow \Pi_1}{\Gamma_1 \Rightarrow \Pi_1}\,\dagger \qquad \dfrac{}{\Sigma \Rightarrow}\,\dagger}{\Gamma \Rightarrow \Pi}\,\star$$

Here the rules marked by † can only be applied above the premises of the rule marked by $\star$ and must share the metavariable Σ. The rules marked by † are unsound by themselves, but the application of $\star$ discharges their occurrences and yield a sound derivation of $\Gamma \Rightarrow \Pi$. The non-locality of systems of rules is twofold: variable sharing constitutes the horizontal dependence between rules occurring in different branches, while rule discharge represents the vertical dependence of the system hierarchy.

In general, a system of rules is a set of possibly labelled sequent rules that are bound to be applied in a predetermined order and that may share schematic variables or labels.

Analyticity of systems of rules when added to a sequent or labelled sequent calculus for classical or intuitionistic logic was proved in [Neg16] for systems acting on atomic formulae or relational atoms.

We consider here a proper restriction of systems of rules: *two-level system of rules*. As explained in Section 3.3.1, the systems in this class correspond to the formulae contained in a superset of the propositional fragment of the class GA_1. Formally, *two-level system of rules* are defined as follows.

Definition 2.3.1. A *two-level system of rules* (*2-system* for short) is a set of sequent rules $\{(r_1), \ldots, (r_k), (r_{bot})\}$ that can only be applied according to the following schema:

$$\dfrac{\begin{array}{ccc}\mathcal{D}_1 & & \mathcal{D}_k \\ \vdots & & \vdots \\ \Gamma \Rightarrow \Pi & \ldots & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\ (r_{bot})$$

where each derivation $\mathcal{D}_i$, for $1 \leq i \leq k$, may contain several applications of

$$\dfrac{\Sigma_1, \Gamma' \Rightarrow \Pi' \quad \ldots \quad \Sigma_n, \Gamma' \Rightarrow \Pi'}{\Sigma_0, \Gamma' \Rightarrow \Pi'}\ (r_i)$$

that act on the same multisets of formulae $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$.

The rule (r_{bot}) is called *bottom rule*, while $(r_1), \ldots, (r_k)$ *top rules*.

Example 2.3.1. The 2-system $Sys_{(\mathrm{com}^*)}$ in [Neg16] for the linearity axiom Lin (cf. Example 2.2.1) is the following (A and B are metavariables for formulae):

$$\dfrac{\dfrac{A, B, \Gamma_1 \Rightarrow \Pi_1}{B, \Gamma_1 \Rightarrow \Pi_1}\ (\mathrm{com}_1^*) \quad \dfrac{A, B, \Gamma_2 \Rightarrow \Pi_2}{A, \Gamma_2 \Rightarrow \Pi_2}\ (\mathrm{com}_2^*)}{\begin{array}{cc}\vdots & \vdots \\ \Gamma \Rightarrow \Pi & \Gamma \Rightarrow \Pi\end{array}}\ (\mathrm{com}_{bot}^*)$$

The derivation of Lin by this rule is the following

$$\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B}{B, A \Rightarrow B}\ (\mathrm{com}_1^*)}{A \Rightarrow B}}{\Rightarrow A \to B}}{\Rightarrow (A \to B) \vee (B \to A)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{A \Rightarrow A}{A, B \Rightarrow A}\ (\mathrm{com}_2^*)}{B \Rightarrow A}}{\Rightarrow B \to A}}{\Rightarrow (A \to B) \vee (B \to A)}}{\Rightarrow (A \to B) \vee (B \to A)}\ (\mathrm{com}_{bot}^*)$$

Notice the similarity with the hypersequent derivation in Example 2.2.1. The same sequents appear in the proof and the rules (com_1^*) and (com_2^*) here have exactly the same rôle as the rule (com) in the hypersequent derivation.

The analyticity of $\mathrm{L}\mathfrak{I} + Sys_{(\mathrm{com}^*)}$ is shown in [Neg16] for *atomic* A and B.

Definition 2.3.1 allows us to apply each rule (r_i) in $\mathcal{D}_i$ several times. Even though this might seem redundant, it is not, as we show in the following example.

Example 2.3.2. A cut-free derivation in $\mathrm{L}\mathfrak{I} + Sys_{(\mathrm{com}^*)}$ (see Example 2.3.1) of the formula $((A \to B) \wedge (A \to B)) \vee ((B \to A) \wedge (B \to A))$ requires two applications of each of the top rules (com_1^*) and (com_2^*):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B}{A, B \Rightarrow B}\,(\mathrm{com}_1^*)}{A \Rightarrow B}}{\Rightarrow A \to B} \quad \dfrac{\dfrac{\dfrac{B \Rightarrow B}{A, B \Rightarrow B}\,(\mathrm{com}_1^*)}{A \Rightarrow B}}{\Rightarrow A \to B}}{\Rightarrow (A \to B) \wedge (A \to B)}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee ((B \to A) \wedge (B \to A))} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \Rightarrow A}{B, A \Rightarrow A}\,(\mathrm{com}_2^*)}{B \Rightarrow A}}{\Rightarrow B \to A} \quad \dfrac{\dfrac{\dfrac{A \Rightarrow A}{B, A \Rightarrow A}\,(\mathrm{com}_2^*)}{B \Rightarrow A}}{\Rightarrow B \to A}}{\Rightarrow (B \to A) \wedge (B \to A)}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee ((B \to A) \wedge (B \to A))}$$

$$\Rightarrow ((A \to B) \wedge (A \to B)) \vee ((B \to A) \wedge (B \to A))$$

2.4 Natural deduction and the Curry–Howard correspondence

Natural deduction has been introduced by Gentzen immediately before sequent calculus in order to provide a formalization of the practice of proving mathematical statements [Gen35]. A natural deduction derivation starts from assumptions, which are formulae temporarily assumed to be true, and proceeds by very simple inference steps that introduce or eliminate logical connectives. Some rules, moreover, can cancel certain assumptions. For example, if assuming A we can prove B:

$$\begin{array}{c} A \\ \vdots \\ B \end{array}$$

then we can infer $A \to B$ from B and cancel the assumption A:

$$\dfrac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \to B}$$

The effect that the implication introduction rule has on the assumption A is clearly non-local and is called *discharge*.

We define now the notion of natural deduction calculus that we will use.

Definition 2.4.1 (Natural deduction calculus)**.** A natural deduction calculus $\mathcal{C}$ is a set of inference rules of the form

$$\dfrac{\begin{array}{ccc}[\mathcal{S}_1] & & [\mathcal{S}_n] \\ \vdots & & \vdots \\ A_1 & \dots & A_n\end{array}}{A}$$

where $\mathcal{S}_1, \dots, \mathcal{S}_n$ are possibly empty sets of formulae.

A natural deduction derivation of a formula A can then be inductively defined as follows:

- A is a derivation of A with open assumption A;
- if

$$\dfrac{\begin{array}{ccc}[\mathcal{S}_1] & & [\mathcal{S}_n] \\ \vdots & & \vdots \\ A_1 & \ldots & A_n\end{array}}{A} \in \mathbb{R}$$

and $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are derivations of $A_1, \ldots, A_n$ with open assumptions $\mathcal{T}_1, \ldots, \mathcal{T}_n$ respectively, then

$$\dfrac{\mathcal{D}_1 \quad \ldots \quad \mathcal{D}_n}{A}$$

is a derivation of A with open assumptions $\bigcup_{i=1}^{n}(\mathcal{T}_i \setminus \mathcal{S}_i)$.

The natural deduction calculus N$\mathfrak{I}$ for intuitionistic logic is defined by the rules in Table 2.4.

$$\dfrac{\bot}{P} \text{ with } P \text{ atomic and } P \neq \bot \qquad \dfrac{A \quad B}{A \wedge B} \qquad \dfrac{A \wedge B}{A} \qquad \dfrac{A \wedge B}{B}$$

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

Table 2.4: Natural deduction calculus N$\mathfrak{I}$.

The table contains two kinds of rules for each binary connective: *introduction rules* and *elimination rules*. We say that a rule *introduces* the connective which is displayed in its conclusion and *eliminates* the connective which is displayed in one of its premises. The premise of an elimination rule in which the connective is displayed is called the *major premise*, all other premises of such a rule are called *minor premises*. The rule for eliminating $\bot$ will be called *ex falso quodlibet*. Notice that the introduction rule of each connective exactly corresponds to the relative description in the BHK interpretation.

As shown in [Pra71], A is a theorem of L$\mathfrak{I}$ if and only if there exists an N$\mathfrak{I}$ derivation of A with no open assumptions. On the other hand, an N$\mathfrak{I}$ derivation of A with open assumptions $A_1, \ldots, A_n$ can be mapped to an L$\mathfrak{I}$ derivation of the sequent $A_1, \ldots, A_n \Rightarrow A$, see [TS96, Section 3.3].

As in the case of sequent and hypersequent calculi, also for natural deduction calculi there are procedures for transforming a generic proof into an analytic one [Pra71]. These procedures are usually called *normalization procedures* and the resulting natural deduction proofs *normal form proofs* or, simply, *normal proofs*.

Notation. Given any natural deduction calculus $\mathcal{C}$, we denote by $\Gamma \vdash_{\mathcal{C}} A$ that there exists a derivation of A in $\mathcal{C}$ with assumptions Γ. Moreover, given a rule or set of rules

$\mathbb{R}$, we denote by $\mathcal{C} + \mathbb{R}$ the calculus obtained by adding the rule $\mathbb{R}$ or the elements of the set $\mathbb{R}$ to $\mathcal{C}$.

As usual, by the *height* of a natural deduction derivation we mean the maximal number of rule applications occurring on any of its branches plus 1.

The requirement that P should be atomic in the *ex falso quodlibet* rule of Table 2.4 is imposed because it simplifies the normalization proofs. We show now that it does not influence the derivability relation defined by the calculus. Indeed we can always derive the general version of the *ex falso quodlibet* rule by constructing a derivation of any formula assuming only $\bot$.

Proposition 2.4.1 (*Ex falso, quodlibet* indeed)**.** *For any formula A, there is a derivation of A in* N$\mathfrak{I}$ *whose only assumption is $\bot$.*

Proof. By induction on the number of occurrences of connectives in A. If A is a propositional variable, we use the rule for $\bot$. If $A = \bot$ the derivation is A itself.

We reason then on the shape of A.

- $A = B \to C$. By induction hypothesis there is an N$\mathfrak{I}$ derivation $\begin{smallmatrix}\mathcal{D}\\ C\end{smallmatrix}$ with only $\bot$ as assumption. We can then obtain the derivation $\dfrac{\begin{smallmatrix}\mathcal{D}\\ C\end{smallmatrix}}{B \to C}$.
- $A = B \wedge C$. By induction hypothesis there are N$\mathfrak{I}$ derivations $\begin{smallmatrix}\mathcal{D}_1\\ B\end{smallmatrix}$ and $\begin{smallmatrix}\mathcal{D}_2\\ C\end{smallmatrix}$ with only $\bot$ as assumption. We can then obtain the derivation $\dfrac{\begin{smallmatrix}\mathcal{D}_1 & \mathcal{D}_2\\ B & C\end{smallmatrix}}{B \wedge C}$.
- $A = B \vee C$. By induction hypothesis there are N$\mathfrak{I}$ derivations $\begin{smallmatrix}\mathcal{D}_1\\ B\end{smallmatrix}$ and $\begin{smallmatrix}\mathcal{D}_2\\ C\end{smallmatrix}$ with only $\bot$ as assumption. We can then obtain either of the derivations $\dfrac{\begin{smallmatrix}\mathcal{D}_1\\ B\end{smallmatrix}}{B \vee C}$ and $\dfrac{\begin{smallmatrix}\mathcal{D}_2\\ C\end{smallmatrix}}{B \vee C}$.

□

2.4.1 Simply typed λ-calculus

The Curry–Howard correspondence formalizes a deep connection between logical systems and functional programming as formalized by λ-calculus. Howard showed that each intuitionistic natural deduction proof of a formula T can be associated with a λ-calculus term t in such a way that the structure of t exactly corresponds to the structure of the proof [How80]. If we consider t as a program, moreover, the formula T specifies the behavior of t in terms of input and output. We say then that T is the type of t and we formally denoted their relation by writing $t : T$.

$$x^A : A \qquad \frac{t : \bot}{t\,\mathsf{efq}_P : P} \quad \text{with } P \text{ atomic and } P \neq \bot$$

$$\frac{t : A \quad u : B}{\langle t, u\rangle : A \wedge B} \qquad \frac{t : A \wedge B}{t\,\pi_0 : A} \qquad \frac{t : A \wedge B}{t\,\pi_1 : B} \qquad \frac{\begin{array}{c}[x^A : A]\\ \vdots\\ t : B\end{array}}{\lambda x^A t : A \to B} \qquad \frac{t : A \to B \quad u : A}{tu : B}$$

where t and u are simply typed λ-terms

Table 2.5: Type assignments for the simply typed λ-calculus.

The typed fragment of λ-calculus including all λ-terms that correspond to some intuitionistic theorem is usually called simply typed λ-calculus, see for example [GLT89]. Constructing a proof of T amounts then to writing a simply typed λ-calculus program t that behaves as specified by T. The connection between intuitionistic natural deduction and simply typed λ-calculus extends further though, indeed the reduction of a natural deduction proof to its normal form corresponds to the evaluation of the associated program.

Table 2.5 contains the type assignment rules for λ-terms. The rules for the types are exactly the rules shown in Table 2.4 for assumptions, $\to$, $\wedge$ and $\bot$. We denote by $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\bot}$ the natural deduction calculus only containing these rules.

Also in $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\bot}$ the general version of the *ex falso quodlibet* rule is derivable.

Proposition 2.4.2 (*Ex falso, quodlibet* indeed)**.** *For any formula A that does not contain $\vee$, there is a derivation of A in $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\bot}$ whose only assumption is $\bot$.*

Proof. The proof of Proposition 2.4.1 trivially specializes to $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\bot}$ for the fragment of intuitionistic logic without disjunction. □

We fix basic notation, definitions and terminology of simply typed λ-calculus [GLT89] that will also apply to the extensions of λ-calculus introduced in Chapters 4, 6, and 5. We denote by $t, u, v \ldots$ the terms which have been typed by the typing rules in Table 2.5 and we call them *proof terms*. The proof terms of simply typed λ-calculus may contain *intuitionistic* variables $x^A_0, x^A_1, x^A_2 \ldots$ of type A for every formula A. These variables are denoted as usual by the metavariables x^A, y^A, $z^A \ldots$ possibly with indices. Whenever the type is not relevant, it will be dropped and we shall simply write $x, y, z \ldots$ Free and bound variables of a proof term are defined as usual. We assume the standard renaming rules and α-equivalences that avoid the capture of variables during reductions.

Notation. By $\langle t_1, t_2, \ldots, t_n\rangle$ we denote the term $\langle t_1, \langle t_2, \ldots \langle t_{n-1}, t_n\rangle \ldots\rangle\rangle$ and by $\langle t_1, t_2, \ldots, t_n\rangle\,\pi_i$, for $i = 1, \ldots, n$, the term $\langle t_1, t_2, \ldots, t_n\rangle\,\pi_1 \ldots \pi_1\pi_0$ containing the projections that select the $(i+1)$th element of the sequence. We define $\langle t_1, \ldots, t_n\rangle$ for $n = 0$ to be $\lambda x^{\bot}\, x^{\bot} : \top$ and we denote the latter as $* : \top$.

The typing rules of simply typed λ-calculus, stripped of λ-terms, are identical to the corresponding inference rules of Gentzen's natural deduction system N$\mathfrak{I}$ for intuitionistic logic. If $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and all free variables of a proof term $t : A$ are in $x_1, \ldots, x_n$, from the logical point of view, t represents an intutionistic natural deduction proof of A from the hypotheses $A_1, \ldots, A_n$. We can then write $\Gamma \vdash t : A$.

The connection between the Curry–Howard correspondence and the BHK interpretation is strikingly close. Indeed, we are simply formalizing the concept of pair by the corresponding programming construct and the informal notion of construction by the notion of function, or program, captured by λ-calculus. The introduction rules directly correspond to the operations specified by the BHK interpretation and the elimination rules follow naturally from them.

As already mentioned, the evaluation of λ-terms can be seen as a procedure for normalizing $\mathrm{N}\mathfrak{I}^{\rightarrow}_{\wedge\perp}$ derivations. For example, λ-calculus β-reduction $(\lambda x^A\, u)t \mapsto u[t/x]$ corresponds to the following proof transformation:

$$\dfrac{\dfrac{\begin{array}{c}[x^A : A]\\ \vdots\\ u : B\end{array}}{\lambda x^A\, u : A \to B} \qquad \begin{array}{c}\vdots\\ t : A\end{array}}{(\lambda x^A\, u)t : B} \quad \mapsto \quad \begin{array}{c}\vdots\\ t : A\\ \vdots\\ u[t/x] : B\end{array}$$

From a proof-theoretical perspective, this transformation is employed to avoid the use of the formula $A \to B$, which might be more complex than necessary and violate the subformula property. The derivation on the left, the one to be reduced, is often called a *redex*. The inference step used to derive $A \to B$, on the other hand, is called a *detour*; it can indeed be considered as an unnecessary deviation from a more direct way of deriving B. The remaining reductions are those for pairs and projections:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ u : A\end{array} \quad \begin{array}{c}\vdots\\ t : B\end{array}}{\langle u, t\rangle : A \wedge B}}{\langle u, t\rangle\, \pi_0 : A} \quad \mapsto \quad \begin{array}{c}\vdots\\ u : A\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\vdots\\ u : A\end{array} \quad \begin{array}{c}\vdots\\ t : B\end{array}}{\langle u, t\rangle : A \wedge B}}{\langle u, t\rangle\, \pi_1 : B} \quad \mapsto \quad \begin{array}{c}\vdots\\ t : B\end{array}$$

Definition 2.4.2 (Normal forms and normalizable terms)**.** As usual, we call *redexes* the terms occurring to the left of $\mapsto_s$ – where the subscript s, if occurs, refers to the relevant system – in the reduction Tables 4.2 and 4.4 in Chapter 4, Table 5.2 in Chapter 5, and Table 6.2 in Chapter 6.

We adopt, for any reduction arrow $\mapsto_s$, the reduction schema: $\mathcal{C}[t] \mapsto_s \mathcal{C}[u]$ whenever $t \mapsto_s u$ and for any context $\mathcal{C}$. Moreover, we denote by $\mapsto_s^*$ the reflexive and transitive closure of the one-step reduction $\mapsto_s$.

A term t is called a *normal form* or, simply, *normal*, if there is no t' such that $t \mapsto_s t'$.

A sequence, finite or infinite, of proof terms $u_1, u_2, \ldots, u_n, \ldots$ is said to be a reduction of t, if $t = u_1$, and for all i, $u_i \mapsto_s u_{i+1}$. A proof term u is *normalizable* if there is a finite reduction of u whose last term is a normal form.

An essential feature that the reductions of a typed calculus should posses is *subject reduction* or, in other terms, the preservation of types. Such property tells us that the reductions do not change the type of the term and hence that they are coherent with the logical system. We are now going to show this result for simply typed λ-calculus, but first we also add the $\vee$ type assignment rules for the injection and case distinction constructs, since we will need them in Chapter 6. The type assignment rules for $\vee$ are

$$\frac{u : A}{\iota_0(u) : A \vee B} \qquad \frac{u : B}{\iota_1(u) : A \vee B} \qquad \frac{\begin{array}{ccc} & [x^A : A] & [y^B : B] \\ & \vdots & \vdots \\ u : A \vee B & w_1 : C & w_2 : C \end{array}}{u\,[x^A.w_1, y^B.w_2] : C}$$

The corresponding reduction rules are

$$\iota_i(t)[x_0.u_0, x_1.u_1] \mapsto u_i[t/x_i] \qquad \text{and} \qquad t[x_0.u_0, x_1.u_1]\xi \mapsto t[x_0.u_0\xi, x_1.u_1\xi]$$

where ξ is a term or π_i for $i \in \{0, 1\}$ or $[y_0.v_0, y_1.v_1]$.

We can now prove subject reduction for simply typed λ-calculus with disjunction.

Theorem 2.4.3 (Subject Reduction)**.** *If $t : A$ and $t \mapsto u$, then $u : A$ and all the free variables of u appear among those of t.*

Proof. 1. $(\lambda x^A\, u)t \mapsto u[t/x]$. Suppose that $(\lambda x^A\, u)t : B$. Since $t : A$ and $x : A$, the term $u[t/x]$ is well defined and, due to the assumptions of the $\to$ introduction type assignment rule, $u[t/x] : B$. Finally, all free variables of t are free in $(\lambda x\, u)t$ as well; and it is easy to prove that the only bound variable of $(\lambda x\, u)t$ which is free in u is x, but x does not occur in $u[t/x]$.

2. $\langle u_0, u_1\rangle\pi_i \mapsto u_i$, for $i = 0, 1$. Suppose that $\langle u_0, u_1\rangle : A_0 \wedge A_1$. By the assumptions of the $\wedge$ introduction type assignment rule, $u_i : A_i$. Moreover, the free variables of u_i are a subset of those of $\langle u_0, u_1\rangle\pi_i$.

3. $\iota_i(t)[x_0^{A_0}.u_0, x_1^{A_1}.u_1] \mapsto u_i[t/x_i]$. Suppose that $\iota_i(t)[x_0.u_0, x_1.u_1] : C$. By the assumptions of the disjunction type assignment rules, $t : A_i$ and $u_i : C$. Hence $u_i[t/x_i]$ is a well defined term of type C. Finally, all free variables of t are free in $\iota_i(t)[x_0^{A_0}.u_0, x_1^{A_1}.u_1]$ as well; and the only bound variable of $\iota_i(t)[x_0^{A_0}.u_0, x_1^{A_1}.u_1]$ which is free in u_i is x_i, but x_i does not occur in $u_i[t/x_i]$.

4. $t[x_0.u_0, x_1.u_1]\xi \mapsto t[x_0.u_0\xi, x_1.u_1\xi]$. Suppose that $t[x_0.u_0, x_1.u_1]\xi : A$ and that $t[x_0.u_0, x_1.u_1] : B$. We have three cases:

   - ξ is a term. Then $\xi : C$ for some C and $B = C \to A$, and $u_i\xi : A$ since $u_i : C \to A$.
   - $\xi = \pi_i$. Then $B = C_0 \wedge C_1$ for some formulae C_0, C_1 such that $A = C_i$, and $u_i\xi : C_i$ since $u_i : C_0 \wedge C_1$.

- $\xi = [y_0.v_0, y_1.v_1]$. Then both $t[x_0.u_0, x_1.u_1]\xi$ and $u_i\xi$ have the same type A.

Hence, in all cases, $t[x_0.u_0\xi, x_1.u_1\xi] : A$. Moreover, all free variables of $t[x_0.u_0\xi, x_1.u_1\xi]$ are free in $t[x_0.u_0, x_1.u_1]\xi$ as well.

□

2.4.2 Beyond simply typed λ-calculus: : the interpretation of classical logic

If we consider the work by Curry and Howard on intuitionistic logic, a natural question arises: can we present similar results for other logics? And if so, do other logics correspond to other notions of computation? Indeed, simply typed λ-calculus directly represents a quite elegant, but also rather simple, model of computation: we cannot represent backtracking or jumps, we cannot include anomalies and mechanisms to handle them, side effects are not allowed, and, more importantly for us, we can only write sequential algorithms. Griffin was the first to answer this question, and he did it by extending Curry–Howard correspondence to classical logic.

Griffin's computational interpretation of classical logic is the typed version of idealized Scheme IS_t [Gri90]. IS_t is a typed version of Felleisen's λ_c, see for example [FFKD86], which extends λ-calculus by an abort operator A and an operator C implementing *call with current continuation* (call/cc). Such programming technique consists in providing functions with a continuation while calling them. Intuitively, such continuation is an abstract representation of the current state of the program.

The operational semantics of these operators is the following:

$$\begin{array}{lcl} \mathcal{E}[\mathsf{A}(s)] & \mapsto & s \\ \mathcal{E}[\mathsf{C}(s)] & \mapsto & s\,\lambda z\,\mathsf{A}(\mathcal{E}[z]) \end{array}$$

The A operator just aborts the computation, represented here by the context $\mathcal{E}[\,]$, and passes the control to its argument, the process s. The C operator passes the control to its argument s as well, but also provides it with a functional abstraction $\lambda z\,\mathsf{A}(\mathcal{E}[z])$ of the context of the computation. Such functional abstraction plays the rôle of the current continuation.

Griffin introduces A and C by the following typing rules:

$$\frac{s : \bot}{\mathsf{A}(s) : F}\ \mathsf{efq} \qquad\qquad \frac{s : \neg\neg F}{\mathsf{C}(s) : F}\ \neg\neg e$$

His motivating argument for such type assignments is the following. Since $\mathcal{E}[\mathsf{C}(s)] \mapsto s\,\lambda z\,\mathsf{A}(\mathcal{E}[z])$, the terms $\mathcal{E}[\mathsf{C}(s)]$ and $s\,\lambda z\,\mathsf{A}(\mathcal{E}[z])$ must have the same type, say G. Now that we know that G is the type of the application of s to $\lambda z\,\mathsf{A}(\mathcal{E}[z])$ we can reconstruct the type of s itself. Indeed, $\mathsf{A}(\mathcal{E}[z]) : G$ because $\mathsf{A}(\mathcal{E}[z]) \mapsto \mathcal{E}[z] : G$. Hence we have that $\lambda z\,\mathsf{A}(\mathcal{E}[z])$ has type $F \to G$ for any type F of z and thus that $s : (F \to G) \to G$. Finally,

if $z : F$ then the occurrence of $[\,]$ in $\mathcal{E}[\,]$ must have type F and therefore $\mathsf{C}(s) : F$ as well. Nevertheless, there is no reason yet for G to be $\bot$. We have a problem, though, if we construct a term of the form $\mathsf{C}(\lambda x\, u)$ where x does not occur in u. The problem with such term is that it can be assigned a type. In fact, it can be assigned any type! Indeed, the type of such term is the same as the type of x, which can be arbitrarily chosen since x does not occur in u. Hence, for the sake of consistency, Griffin allows only type assignments in which G is $\bot$, thus obtaining a type assignment rule corresponding to the double negation elimination $((F \to \bot) \to \bot) \to F$.

The reduction rule for the C operator corresponds to the following proof transformation:

$$\dfrac{\begin{array}{c}\vdots\\ s : \neg\neg F\end{array}}{\begin{array}{c}\mathsf{C}(s) : F\\ \vdots\\ \mathcal{E}[\mathsf{C}(s)] : \bot\end{array}} \quad \mapsto \quad \dfrac{\begin{array}{c}\vdots\\ s : \neg\neg F\end{array} \quad \dfrac{\begin{array}{c}[z : F]\\ \vdots\\ \mathcal{E}[z] : \bot\end{array}}{\mathsf{A}(\mathcal{E}[z]) : \bot}}{\lambda z\, \mathsf{A}(\mathcal{E}[z]) : \neg F}}{s\, \lambda z\, \mathsf{A}(\mathcal{E}[z]) : \bot}$$

This transformation is admissible only when the type of $\mathcal{E}[\,]$ is $\bot$, otherwise it would not be possible to obtain a term of type $\neg F$ abstracting $\mathcal{E}[\,]$. Notice that the introduction of A does not have any logical meaning and is simply needed to match the term reduction.

A second milestone in the history of computational interpretations of classical logic is Parigot's λ_μ-calculus [Par92]. Such calculus is based on a multiple-conclusion natural deduction formalism called *free deduction*. Such system generalizes natural deduction style rules to multisets of formulae. For example the free deduction implication introduction rule, expressed in our notation, is

$$\dfrac{\begin{array}{c}[A]\\ \vdots\\ \Delta, B\end{array}}{\Delta, A \to B}$$

we just add a context Δ to the usual rule. In order to keep track of context and active formulae we can use labels: we adopt the convention that we can apply introduction and elimination rules only to unlabeled formulae and we introduce suitable rules to add or remove labels.

If we project this idea onto λ-terms, we have λ_μ. When we label a formula by a name β, we also name the term β (below left); when we remove a label β form a formula, we bind that name in the term using a μ operator (below right).

$$\dfrac{t : A, \Delta}{[\beta]t : A^\beta, \Delta} \qquad\qquad \dfrac{t : A^\beta, \Delta}{\mu\beta\, t : A, \Delta}$$

Now, names and μ give us a complete map of the history of a formula: $\mu\beta$ means that the formula is about to be used in a rule application, $[\beta]$ means that the formula have just been produced by a rule application.

This solves the normalization issues that the multiple conclusion setting raises. For example, while removing an implication redex we might not find the implication introduction just above its elimination because the implication, just before being eliminated, was part of the context (bottom part of the derivation below left). But this is not a problem: we just need to find the rule that put the implication in the context and move the redex above that rule (below right).

$$\dfrac{\dfrac{\dfrac{\vdots}{w : A \to B, \Delta'}}{[\beta]w : (A \to B)^\beta, \Delta'}\quad \vdots}{\dfrac{\dfrac{u : (A \to B)^\beta, \Delta}{\mu\beta.u : A \to B, \Delta} \qquad \vdots\ v : A, \Sigma}{(\mu\beta.u)\, v : B, \Delta, \Sigma}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\vdots}{w : A \to B, \Delta'} \quad \vdots\ v : A, \Sigma}{w\, v : B, \Delta', \Sigma}}{\dfrac{[\beta]w\, v : (B)^\beta, \Delta', \Sigma}{\vdots}}\quad \dfrac{u[[\beta](wv)/[\beta]w] : (B)^\beta, \Delta, \Sigma}{\mu\beta.u[[\beta](wv)/[\beta]w] : B, \Delta, \Sigma}$$

The jump of the implication elimination rule exactly corresponds to the jump of the argument v of the term $\mu\beta\, u$ in $(\mu\beta.u)\, v \quad \mapsto \quad \mu\beta.u[[\beta](wv)/[\beta]w]$.

From a programming perspective, this kind of jumps corresponds to some extent to a *goto* instruction. Moreover, the behavior of control operators, such as call/cc, can be simulated using it.

The last computational interpretation of classical logic that we will consider here is de Groote's λ_{exn} [dG95], which shares with λ_{Cl} the natural deduction calculus $\mathrm{N}\mathfrak{Cl}$. It is therefore particularly interesting for us.

The calculus λ_{exn} provides a type system for a pair of operators for raising and handling exceptions. *Exceptions* are anomalous conditions occurring during a computation that hinder the normal execution of the program and thus require special processing. Errors are a typical example. When such anomalous conditions occur, an exception is *raised* and the required information about the anomaly is sent to a program that processes it or, technically, *handles the exception*. The calculus λ_{exn} contains the operators

$$(\mathsf{raise}\, y\, s) \qquad \text{and} \qquad \mathsf{let}\ y : \neg A\ \mathsf{in}\ u\ \mathsf{handle}\ (y\, x)\ \Rightarrow\ t\ \mathsf{end}$$

for raising and handling exceptions, respectively. When an error, represented by a term of type $\bot$, occurs, the λ_{exn} term can raise the exception above on the left containing the information s. If the exception occurs as u inside the term above on the right, then the content s of the exception is used by t to handle the exception, as in the reduction

$$\mathsf{let}\ y : \neg A\ \mathsf{in}\ (\mathsf{raise}\, y\, s)\ \mathsf{handle}\ (y\, x)\ \Rightarrow\ t\ \mathsf{end} \quad\mapsto\quad t[s/x]$$

A term $(\mathsf{raise}\, y\, s)$ in λ_{exn} is typed by the *ex falso quodlibet* rule: a proof of $\bot$ is interpreted as a program incurring in an error, and using that occurrence of $\bot$ to derive another formula is interpreted as raising an exception for such error:

$$\dfrac{u : \bot}{(\mathsf{raise}\, u) : B}$$

The operator for handling exceptions on the other hand, is typed by the rule for the excluded middle axiom $\neg A \vee A$ that is also used for λ_{Cl}:

$$\frac{\begin{array}{cc}[y : \neg A] & [x : A] \\ \vdots & \vdots \\ v : B & t : B\end{array}}{\mathsf{let}\ y : \neg A\ \mathsf{in}\ v\ \mathsf{handle}\ (y\,x)\ \Rightarrow\ t\ \mathsf{end} : B}$$

A term typed by this rule has the following computational meaning: if the term v raises an exception, the term t is supposed to handle it. If we have the configuration below on the left inside a proof term, the exception $(\mathsf{raise}\, y\, s)$ is handled, as the reduction shows, by substituting the content s of such exception for the variable x in the term t:

$$\frac{\dfrac{\dfrac{\vdots}{[y : \neg A] \quad s : A}}{y\, s : \bot}}{(\mathsf{raise}\, y\, s) : B} \qquad \begin{array}{c}[x : A] \\ \vdots \\ t : B\end{array}}{\mathsf{let}\ y : \neg A\ \mathsf{in}\ (\mathsf{raise}\, y\, s)\ \mathsf{handle}\ (y\,x)\ \Rightarrow\ t\ \mathsf{end} : B} \qquad \mapsto \qquad \begin{array}{c}\vdots \\ s : A \\ \vdots \\ t : B\end{array}$$

The calculus λ_{exn} contains a more general version of this reduction, but this is enough here to understand the basic ideas on which such calculus is based. From a proof-theoretical point of view, this is an elimination of the excluded middle instances whose $\neg A$ assumptions are only used with a proof of A to derive $\bot$. Indeed, if there is a proof of A, we can simply use it to derive the assumption A needed to prove B in the right branch of the excluded middle rule.

By introducing the calculus λ_{Cl} in Section 4.1.1 we provide a computational interpretation in terms of parallel computation of the logical system of λ_{exn}. From a logical point of view, a crucial difference between the two systems lies in the fact that the normalization procedure of λ_{Cl}, unlike that of λ_{exn}, yields normal proof terms that enjoy the subformula property.

2.4.3 Avron's thesis and natural deduction calculi for intermediate logics

The connection between concurrent computation and hypersequent calculi was first noted by Avron. His 1991 thesis states, in particular, that it should be possible to use hypersequent calculi "as bases for parallel λ-calculi" [Avr91]. As discussed in Section 2.4.1, the most established way to associate a λ-calculus to a proof calculus is by a Curry–Howard correspondence for a natural deduction calculus. The absence of explicit structural elements makes it possible indeed to use formulae as types of λ-terms, rather than complex objects such as sequents. Hence, in order to confirm Avron's thesis, some work has been devoted in finding a way to represent hypersequent proofs as natural deduction proofs.

Prominent examples of natural deduction calculi inspired by hypersequent calculi are [BCF00] and [BP15]. The formalisms adopted in both papers rely on quite heavy

structural generalizations of the natural deduction formalism. Both are indeed based on the idea of joining together natural deduction derivations by mechanisms or explicit operators similar to the structural connective | of hypersequents.

If we follow Avron's intuition [Avr91] it seems natural to try to introduce a parallelism operator directly between natural deduction derivations. A typical hypersequent derivation seems indeed to be composed of different parallel sequent proofs. For example we could say that the hypersequent derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B \quad C, A \Rightarrow A}{A \Rightarrow B \mid C, B \Rightarrow A}\,(\text{com}) \quad \dfrac{C \Rightarrow C \quad A, B \Rightarrow A}{A \Rightarrow C \mid C, B \Rightarrow A}\,(\text{com})}{A \Rightarrow B \wedge C \mid C, B \Rightarrow A}}{A \Rightarrow B \wedge C \mid C \wedge B \Rightarrow A}}{A \Rightarrow B \wedge C \mid\Rightarrow C \wedge B \to A}}{\Rightarrow A \to B \wedge C \mid\Rightarrow C \wedge B \to A}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A) \mid\Rightarrow C \wedge B \to A}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A) \mid\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)}$$

consist of the two parallel proofs

$$\dfrac{\dfrac{\dfrac{\dfrac{A \Rightarrow B \quad A \Rightarrow C}{A \Rightarrow B \wedge C}}{A \Rightarrow B \wedge C}}{A \Rightarrow B \wedge C}}{\Rightarrow A \to B \wedge C}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)} \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{\dfrac{C, B \Rightarrow A \quad C, B \Rightarrow A}{C, B \Rightarrow A}}{C \wedge B \Rightarrow A}}{\Rightarrow C \wedge B \to A}}{\Rightarrow C \wedge B \to A}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)}$$

The applications of the (com) rule are then used to exchange occurrences of A for occurrences of B or C. This intuition is exploited in Chapter 3 to show the equivalence of hypersequents and 2-systems. Some trivial inference step is employed in the two derivations above, but they are clearly equivalent to the following sequent derivations:

$$\dfrac{\dfrac{\dfrac{A \Rightarrow B \quad A \Rightarrow C}{A \Rightarrow B \wedge C}}{\Rightarrow A \to (B \wedge C)}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)} \qquad \dfrac{\dfrac{\dfrac{C, B \Rightarrow A}{C \wedge B \Rightarrow A}}{\Rightarrow C \wedge B \to A}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)}$$

Following these considerations, and since the correspondence between natural deduction and sequent calculus is well established, it is perfectly natural to define hyper natural deductions introducing an operator that joins different natural deduction derivations. This is the direction taken by the approaches of [BCF00] and [BP15]. It turns out, though, that there is no need to explicitly introduce such an operator. The formalism of systems of rules shows very clearly that it is possible to rely only on non-local dependencies to establish a connection between different derivations. In particular, if we impose conditions which connect the rules that execute the exchange of formulae, there is no need for new

structural elements. For sequents, the result is

$$\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B}{A \Rightarrow B}\,{*} \quad \dfrac{C \Rightarrow C}{A \Rightarrow C}\,{**}}{A \Rightarrow B \wedge C}}{\Rightarrow A \to (B \wedge C)}}{\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)} \qquad \dfrac{\dfrac{\dfrac{C, A \Rightarrow A}{C, B \Rightarrow A}\,{*}}{C \wedge B \Rightarrow A}}{\Rightarrow C \wedge B \to A} \qquad \dfrac{\dfrac{\dfrac{A, B \Rightarrow A}{C, B \Rightarrow A}\,{**}}{C \wedge B \Rightarrow A}}{\Rightarrow C \wedge B \to A}$$

$$\Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A) \qquad \Rightarrow (A \to B \wedge C) \vee (C \wedge B \to A)$$

We duplicate here the derivation on the right in order to apply the two rules marked by $*$ and the two rules marked by $**$. Instead of an application of the hypersequent rule (com) such as

$$\dfrac{B \Rightarrow B \quad C, A \Rightarrow A}{A \Rightarrow B \mid C, B \Rightarrow A}\ (\text{com})$$

we have now a pair of related sequent rule applications:

$$\dfrac{B \Rightarrow B}{A \Rightarrow B}\,{*} \qquad\qquad \dfrac{C, A \Rightarrow A}{C, B \Rightarrow A}\,{*}$$

In natural deduction it is even simpler since the result

$$\dfrac{\dfrac{\dfrac{\dfrac{[A]^1}{B}\,{*} \quad \dfrac{[A]^1}{C}\,{**}}{B \wedge C}}{A \to (B \wedge C)}\,1}{(A \to B \wedge C) \vee (C \wedge B \to A)} \qquad \dfrac{\dfrac{\dfrac{[C \wedge B]^2}{B}\,{*}}{A}}{C \wedge B \to A}\,2 \qquad \dfrac{\dfrac{\dfrac{[C \wedge B]^3}{C}\,{**}}{A}}{C \wedge B \to A}\,3$$

$$(A \to B \wedge C) \vee (C \wedge B \to A) \qquad (A \to B \wedge C) \vee (C \wedge B \to A)$$

directly corresponds to a completely traditional natural deduction derivation using some instances of the Lin axiom as assumptions:

$$\dfrac{(A \to B) \vee (B \to A) \qquad \dfrac{(A \to C) \vee (C \to A) \qquad \dfrac{\dfrac{\dfrac{[A \to B] \quad [A]^1}{B} \quad \dfrac{[A \to C] \quad [A]^1}{C}}{B \wedge C}\,1}{A \to (B \wedge C)}\,\alpha \qquad \dfrac{\dfrac{[C \to A] \quad \dfrac{[C \wedge B]^3}{C}}{A}\,3}{C \wedge B \to A}\,\alpha}{\alpha} \qquad \dfrac{\dfrac{[B \to A] \quad \dfrac{[C \wedge B]^2}{B}}{A}\,2}{C \wedge B \to A}\,\alpha}{\alpha}$$

$$(A \to B \wedge C) \vee (C \wedge B \to A)$$

where $\alpha = (A \to B \wedge C) \vee (C \wedge B \to A)$.

CHAPTER 3

From hypersequent calculi to natural deduction

In order to define Curry–Howard correspondences for intermediate logics that are naturally characterized by hypersequent calculi, we first establish a connection between hypersequents and natural deduction. We achieve this by defining a formal correspondence between the hypersequent formalism and a formalism which extends sequent calculus by non-local mechanisms similar to natural deduction assumption discharge: systems of rules. The strong similarity between systems of rules and natural deduction will then enable us to automatically define natural deduction calculi corresponding to hypersequent calculi and to transfer the intuitions coming from the study of hypersequents to the study of the obtained natural deduction calculi.

Formally, we show that the calculi in a rather simple fragment of systems of rules are equivalent to hypersequent calculi and can be directly used to define simple and modular natural deduction calculi retaining the parallel structure of hypersequents for a large class of intermediate logics. The calculi are obtained by extending N$\mathfrak{I}$ by new rules. Similarly to sequent rules belonging to systems, these rules can discharge other rule applications, or *higher-level rules* according to the terminology of [SH14]. Such natural deduction rules are the base of the Curry–Howard correspondences presented in Chapters 4, 5, and 6.

A possible connection between hypersequents and systems of rules is hinted in [Neg16]. We formalize and prove this intuition. Focusing on propositional intermediate logics, we define a bi-directional embedding between hypersequents and a subclass of non-labelled systems of rules (*2-systems*) in which the vertical non-locality is restricted to at most two sequent rules. The embeddings show that these two seemingly different extensions of the sequent calculus have the same expressive power.

The embedding of 2-systems into hypersequents provides furthermore very general analyticity results for 2-system calculi. Analyticity results are shown in [Neg16] for

systems of rules sharing only first-order variables or atomic formulae, and while this restriction does not yield any loss of generality for labelled sequents, it does for systems of rules directly corresponding to Hilbert axioms. The embedding also enables us to introduce novel cut-free 2-systems.

The present chapter is based on [CG16] and [CG18], and structured as follows. The translations between systems of rules and hypersequent rules are presented in Section 3.1; Section 3.2 contains the embeddings between derivations: Section 3.2.1 the direction from system of rules to hypersequent derivations and Section 3.2.2 the inverse direction. Section 3.3, finally, shows the applications of the embedding, which include the definition of new natural deduction calculi for a large class of intermediate logics. The latter result will be the base of the work presented in the subsequent chapters.

3.1 From 2-systems to hypersequent rules and back

We show how to rewrite a 2-system *Sys* into the corresponding hypersequent rule Hr_{Sys}; vice versa, from a hypersequent rule *Hr* we construct the corresponding 2-system Sys_{Hr}. The transformation of derivations from $\mathrm{H}\mathfrak{I} + Hr$ into $\mathrm{L}\mathfrak{I} + Sys_{Hr}$ (and from $\mathrm{L}\mathfrak{I} + Sys$ into $\mathrm{H}\mathfrak{I} + Hr_{Sys}$) is shown in Section 3.2.

From 2-systems to hypersequent rules

Given a 2-system *Sys* of the form

$$\dfrac{\begin{array}{ccc}\mathcal{D}_1 & & \mathcal{D}_k \\ \vdots & & \vdots \\ \Gamma \Rightarrow \Pi & \dots & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\ (r_{bot})$$

where each derivation $\mathcal{D}_i$, for $1 \leq i \leq k$, may contain several applications of the rule

$$\dfrac{A_i^1, \dots, A_i^{l_i}, \Gamma_i \Rightarrow \Pi_i \quad \dots \quad B_i^1, \dots, B_i^{m_i}, \Gamma_i \Rightarrow \Pi_i}{C_i^1, \dots, C_i^{n_i}, \Gamma_i \Rightarrow \Pi_i}\ (r_i)$$

the corresponding hypersequent rule Hr_{Sys} is as follows:

$$\dfrac{M_1 \qquad \dots \qquad M_k}{G \mid C_1^1, \dots, C_1^{n_1}, \Gamma_1 \Rightarrow \Pi_1 \mid \dots \mid C_k^1, \dots, C_k^{n_k}, \Gamma_k \Rightarrow \Pi_k}$$

where M_i, for $1 \leq i \leq k$, is the multiset of premises

$$G \mid A_i^1, \dots, A_i^{l_i}, \Gamma_i \Rightarrow \Pi_i \quad \dots \quad G \mid B_i^1, \dots, B_i^{m_i}, \Gamma_i \Rightarrow \Pi_i$$

Example 3.1.1. From Negri's 2-system in Example 2.3.1 we obtain the rule acting on formulae A, B

$$\dfrac{G \mid A, B, \Gamma_1 \Rightarrow \Pi_1 \quad G \mid A, B, \Gamma_2 \Rightarrow \Pi_2}{G \mid B, \Gamma_1 \Rightarrow \Pi_1 \mid A, \Gamma_2 \Rightarrow \Pi_2}\ (\mathrm{com}^*)$$

From hypersequent rules to 2-systems

Given any hypersequent rule Hr of the form

$$\frac{M_1 \qquad \dots \qquad M_k}{G \mid \Theta_1^1, \dots, \Theta_1^{n_1}, \Gamma_1 \Rightarrow \Pi_1 \mid \dots \mid \Theta_k^1, \dots, \Theta_k^{n_k}, \Gamma_k \Rightarrow \Pi_k}$$

where the sets M_i, for $1 \leq i \leq k$, constitute a partition of the set of premises of Hr and each M_i contains the premises

$$G \mid S_i^1 \quad \dots \quad G \mid S_i^{m_i}$$

where $S_i^1, \dots, S_i^{m_i}$ are sequents. The corresponding 2-system Sys_{Hr} is

$$\frac{\begin{array}{ccc}\mathcal{D}_1 & & \mathcal{D}_k \\ \vdots & & \vdots \\ \Gamma \Rightarrow \Pi & \dots & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\ (r_{bot})$$

where the derivation $\mathcal{D}_i$, for $1 \leq i \leq k$, may contain several applications of the rule

$$\frac{S_i^1 \quad \dots \quad S_i^{m_i}}{\Theta_i^1, \dots, \Theta_i^{n_i}, \Gamma_i \Rightarrow \Pi_i}\ (r_i)$$

Definition 3.1.1. We say that the premises of Hr contained in M_i, for $1 \leq i < k$, are *linked* to the component $\Theta_i^1, \dots, \Theta_i^{n_1}, \Gamma_i \Rightarrow \Pi_i$ of the conclusion.

Example 3.1.2. The rewriting $Sys_{(\mathrm{com})}$ of the rule (com) in Example 2.2.1 is

$$\frac{\dfrac{\Sigma, \Gamma_1 \Rightarrow \Pi_1}{\Delta, \Gamma_1 \Rightarrow \Pi_1}\ (\mathrm{com}_1) \qquad \dfrac{\Delta, \Gamma_2 \Rightarrow \Pi_2}{\Sigma, \Gamma_2 \Rightarrow \Pi_2}\ (\mathrm{com}_2)}{\Gamma \Rightarrow \Pi}\ (\mathrm{com}_{bot})$$

3.2 Embedding the two formalisms

We introduce algorithms for transforming 2-system derivations into hypersequent derivations and vice versa.

3.2.1 From 2-systems to hypersequent derivations

Consider any set $\mathbb{SR}$ of 2-systems and set $\mathbb{HR}$ of hypersequent rules s.t. if $Sys \in \mathbb{SR}$ then $Hr_{Sys} \in \mathbb{HR}$. Starting from a derivation $\mathcal{D}$ in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ we construct a derivation $\mathcal{D}'$ in $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ of the same end-sequent. The construction proceeds by a stepwise translation of the rules in $\mathcal{D}$: the rules of $\mathrm{L}\mathfrak{I}$ are translated into rules of $\mathrm{H}\mathfrak{I}$ – possibly using (EW); the top rules of 2-systems in $\mathbb{SR}$ are translated into applications of the corresponding rules

in $\mathbb{HR}$ – and additional (EW), if needed; and the relative bottom rules are translated into applications of (EC). To keep track of the various translation steps, we mark the derivation $\mathcal{D}$. We start by marking and translating the leaves of $\mathcal{D}$. The rules with marked premises are then translated one by one and the marks are moved to the conclusions of the rules. The process is repeated until we reach and translate the root of $\mathcal{D}$. The correct termination of the procedure is guaranteed when $\mathcal{D}$ satisfies the following conditions:

1. two applications of a top rule belonging to the same 2-system instance never occur on the same path of the derivation,

2. for each pair of 2-system instances, no top rule of one of the two instances occurs below any top rule of the other instance (see Definition 3.2.1 as used in Lemma 3.2.3)

Section 3.2.1 shows that each 2-system derivation can be transformed into one satisfying them.

The algorithm

Input: a derivation $\mathcal{D}$ in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$. Output: a derivation $\mathcal{D}'$ of the same sequent in $\mathrm{H}\mathfrak{I} + \mathbb{HR}$.

Translating axioms. The leaves of $\mathcal{D}$ are marked and copied as leaves of $\mathcal{D}'$.

Translating rules. Rules are translated one by one in the following order: first the one-premise logical and structural rules applied to marked sequents, then the two-premise logical rules and bottom rules with all premises marked, and finally all the top rules of one 2-system instance[1]. After having translated each rule – or all top rules of a 2-system instance – we remove the marks from the premises of the translated rules and mark their conclusions.

When we translate the top rules of a 2-system we apply the corresponding hypersequent rule once for each possible combination of different top rules of such system. For instance, if a 2-system contains two applications $(r_1)'$ and $(r_1)''$ of one top rule, and one application (r_2) of another top rule, we will have one hypersequent rule application translating the pair $((r_1)', (r_2))$, and one hypersequent rule application translating the pair $((r_1)'', (r_2))$.

Since the $\mathrm{L}\mathfrak{I}$ rules are particular instances of $\mathrm{H}\mathfrak{I}$ rules, we only show how to translate 2-systems. Hence, consider a 2-system $Sys \in \mathbb{SR}$ applied in $\mathcal{D}$ with the following instances of

[1] Condition 1 above guarantees that all top rules of a 2-system instance can be translated by one hypersequent rule.

1. top rules:

$$\frac{\begin{array}{ccc}\vdots & & \vdots\\ S_1^1 & \dots & S_1^{m_1}\end{array}}{\Delta_1, \Gamma_1 \Rightarrow \Pi_1}\ (r_1) \qquad \dots \qquad \frac{\begin{array}{ccc}\vdots & & \vdots\\ S_k^1 & \dots & S_k^{m_k}\end{array}}{\Delta_k, \Gamma_k \Rightarrow \Pi_k}\ (r_k)$$

   where $S_1^1, \dots, S_1^{m_1}, \dots, S_k^1, \dots, S_k^{m_k}$ are marked sequents and each top rule $(r_1), \dots, (r_k)$ is possibly applied more than once.

   By the definition of the algorithm, we have hypersequent derivations of

$$G \mid S_1^1 \quad \dots \quad G \mid S_1^{m_1} \qquad \dots \qquad G \mid S_k^1 \quad \dots \quad G \mid S_k^{m_k}$$

   for each application of the top rules. We apply Hr_{Sys} as follows

$$\frac{M_1 \qquad \dots \qquad M_k}{G \mid \Delta_1, \Gamma_1 \Rightarrow \Pi_1 \mid \dots \mid \Delta_k, \Gamma_k \Rightarrow \Pi_k}$$

   for each possible combination of k applications of the top rules $(r_1), \dots, (r_k)$ – possibly duplicating the hypersequent derivations previously obtained. We move the marks to the conclusions of $(r_1), \dots, (r_k)$.

   Notice that we always have hypersequents containing suitable active components and matching context components. Indeed, given that we translate into a hypersequent rule application each possible combination of top rules, at each translation step (above the bottom rule) we have exactly one hypersequent for each possible combination of marked sequents.

2. bottom rule:

$$\frac{\begin{array}{cc}\vdots & \vdots\\ \Gamma \Rightarrow \Pi & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\ (r_{bot})$$

   Without loss of generality we can assume that the top rules of the considered 2-system have been applied above the premises of (r_{bot}) – as otherwise the application of the 2-system is redundant. Hence we have a derivation in $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ of $G \mid \Gamma \Rightarrow \Pi \mid \dots \mid \Gamma \Rightarrow \Pi$. The desired derivation of $G \mid \Gamma \Rightarrow \Pi$ is obtained by repeatedly applying (EC). We move the marks to the conclusion of (r_{bot}).

Theorem 3.2.1. *For any set $\mathbb{HR}$ of hypersequent rules and set $\mathbb{SR}$ of 2-systems s.t. if $Sys \in \mathbb{SR}$ then $Hr_{Sys} \in \mathbb{HR}$, if $\vdash_{\mathrm{L}\mathfrak{I}+\mathbb{SR}} \Gamma \Rightarrow \Pi$ then $\vdash_{\mathrm{H}\mathfrak{I}+\mathbb{HR}} \Gamma \Rightarrow \Pi$.*

Proof. Apply the above algorithm to the $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ derivation $\mathcal{D}$ of $\Gamma \Rightarrow \Pi$ to obtain $\mathcal{D}'$. The algorithm terminates because the number of rule applications in a derivation is finite. We show that the algorithm does not stop before translating the root of $\mathcal{D}$. The proof is by induction on the number u of 2-system instances whose top rules are still to be translated. If $u = 0$ all remaining rules can be translated as soon as the premises are marked. Assume $u = n + 1$. As we will show in Lemma 3.2.3, see also Definition 3.2.1,

there is at least a 2-system instance S whose top rules are still untranslated and do not occur below any untranslated top rule. Hence the rule applications that have to be translated before the top rules of S do not belong to any 2-system and can be translated as soon as their premises are marked. After translating these rules, we can translate the top rules of S and obtain $u = n$. □

Example 3.2.1. The following derivation in the calculus $\mathrm{L}\mathfrak{I} + Sys_{(\mathrm{com})}$ for Gödel logic (see Example 3.1.2)

$$\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B}{A \Rightarrow B}\,(\mathrm{com}_1)'}{\Rightarrow A \to B} \quad \dfrac{\dfrac{B \Rightarrow B}{A \Rightarrow B}\,(\mathrm{com}_1)''}{\Rightarrow A \to B}}{\Rightarrow (A \to B) \wedge (A \to B)}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A)} \qquad \dfrac{\dfrac{\dfrac{A \Rightarrow A}{B \Rightarrow A}\,(\mathrm{com}_2)}{\Rightarrow B \to A}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A)}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A)}\,(\mathrm{com}_{bot})$$

is translated into the $\mathrm{H}\mathfrak{I} + (\mathrm{com})$ derivation (see Example 2.2.1)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B \Rightarrow B \quad A \Rightarrow A}{A \Rightarrow B \mid B \Rightarrow A}\,(\mathrm{com})'}{A \Rightarrow B \mid\Rightarrow B \to A}}{\Rightarrow A \to B \mid\Rightarrow B \to A} \quad \dfrac{\dfrac{\dfrac{B \Rightarrow B \quad A \Rightarrow A}{A \Rightarrow B \mid B \Rightarrow A}\,(\mathrm{com})''}{A \Rightarrow B \mid\Rightarrow B \to A}}{\Rightarrow A \to B \mid\Rightarrow B \to A}}{\Rightarrow (A \to B) \wedge (A \to B) \mid\Rightarrow B \to A}}{\Rightarrow (A \to B) \wedge (A \to B) \mid\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A)}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A) \mid\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A)}}{\Rightarrow ((A \to B) \wedge (A \to B)) \vee (B \to A)}\,(\mathrm{EC})$$

where $(\mathrm{com})'$ translates the pair of top rule applications $((\mathrm{com}_1)', (\mathrm{com}_2))$, while $(\mathrm{com})''$ translates the pair $((\mathrm{com}_1)'', (\mathrm{com}_2))$.

Normal forms of 2-systems derivations

We introduce the normal forms of 2-system derivations required by the algorithm of Section 3.2.1 and we show how to obtain them. The definition of 2-systems (Definition 2.3.1) is indeed decidedly liberal. It allows unrestricted nesting of 2-systems and does not limit in any way the application of the top rule (r_i) inside $\mathcal{D}_i$. Such freedom matches naturally the general idea of a system of rules, but complicates the structure of derivations and the algorithm for transforming 2-system derivations into hypersequent derivations. We show below that we can consider, without loss of generality, derivations of a simplified form.

Lemma 3.2.2. *Any 2-system derivation can be transformed into one with the following property: two applications of a top rule (t) belonging to the same 2-system instance never occur on the same path of the derivation.*

Proof. Consider any 2-system derivation in which two applications of (t) occur along the same path:

$$\dfrac{\dfrac{\vdots}{\Sigma_1, \Gamma' \Rightarrow \Pi'} \quad \ldots \quad \dfrac{\vdots}{\Sigma_n, \Gamma' \Rightarrow \Pi'}\,(t)}{\Delta, \Gamma' \Rightarrow \Pi'} \qquad \dfrac{\begin{array}{ccccc}\vdots & & \vdots\ \mathcal{D} & & \vdots \\ \Sigma_1, \Gamma \Rightarrow \Pi & \ldots & \Sigma_i, \Gamma \Rightarrow \Pi & \ldots & \Sigma_n, \Gamma \Rightarrow \Pi\end{array}\,(t)}{\Delta, \Gamma \Rightarrow \Pi}$$

We use (IW) and (IC) to transform it into

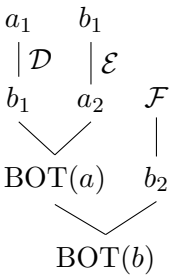
$$\dfrac{\begin{array}{ccccc}\vdots & & \dfrac{\vdots}{\Sigma_i, \Gamma' \Rightarrow \Pi'}\,(\mathrm{IW}) & & \vdots \\ \Sigma_1, \Gamma \Rightarrow \Pi & \ldots & \dfrac{\vdots\ \mathcal{D}'}{\Sigma_i, \Sigma_i, \Gamma \Rightarrow \Pi}\,(\mathrm{IC}) & \ldots & \Sigma_n, \Gamma \Rightarrow \Pi\end{array}\,(t)}{\Delta, \Gamma \Rightarrow \Pi}$$

where for each sequent $\Gamma'' \Rightarrow \Pi''$ in $\mathcal{D}$ there is a sequent $\Sigma_i, \Gamma'' \Rightarrow \Pi''$ in $\mathcal{D}'$. □

Derivations using 2-systems can be further simplified. Indeed the lemma below shows that we can restrict our attention to derivations with a limited nesting of 2-systems. We use the notion of entanglement to formalize a violation of this limitation.

Definition 3.2.1. Two 2-system instances S_1 and S_2 are *entangled* if some top rules of S_1 occur above some top rules of S_2 and some of the former occur below some of the latter.

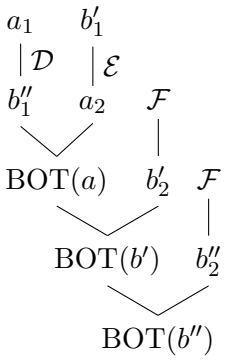
Consider, for instance, the following derivation schema containing two 2-system instances a and b with bottom rules BOT(a) and BOT(b) and top rules a_1, a_2 and b_1, b_2, respectively:



We use $\mathcal{D}$, $\mathcal{E}$ and $\mathcal{F}$ to denote derivations. The entanglement here occurs because b_1 is applied once below a_1 and once above a_2.

Notice that a pair of 2-systems can be entangled only if their bottom rules occur one above the other. Thus, all rules of one of them necessarily occur above one premise of the bottom rule of the other.

Example 3.2.2. To disentangle a and b, we make two copies b' and b'' of b that are going to contain the rules formerly belonging to b:



The 2-system instances are now disentangled: no top rule of b' occurs below any top rule of a and no top rule of b'' occurs above any top rule of a.

Such transformation is an example of the basic step employed in the following lemma.

Lemma 3.2.3. *Any 2-system derivation $\mathcal{P}$ can be transformed into a 2-system derivation $\mathcal{P}'$ of the same end-sequent in which there is no pair of entangled 2-system instances.*

Proof. First we introduce a transformation of derivations (*e-reduction*) that reduces the number of top rule applications involved in entanglements. Then we provide a strategy to obtain the desired derivation $\mathcal{P}'$ using such transformation, and we prove termination.

E-reduction: given a 2-system instance $\mathcal{S}$ with bottom rule ($B_{\mathcal{S}}$) entangled with 2-system instances $\mathcal{S}_1, \ldots, \mathcal{S}_n$:

$$\dfrac{\begin{array}{ccc}\mathcal{D}_1 & & \mathcal{D}_n \\ \vdots & & \vdots \\ \Gamma \Rightarrow \Pi & \ldots & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\,(B_{\mathcal{S}})$$

we make two copies $\mathcal{S}'$ and $\mathcal{S}''$ of $\mathcal{S}$ with bottom rules ($B_{\mathcal{S}'}$) respectively ($B_{\mathcal{S}''}$):

$$\dfrac{\dfrac{\begin{array}{ccccc}\mathcal{D}'_1 & & \mathcal{D}_2 & & \mathcal{D}_m \\ \vdots & & \vdots & & \vdots \\ \Gamma \Rightarrow \Pi & \Gamma \Rightarrow \Pi & \ldots & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\,(B_{\mathcal{S}'}) \qquad \begin{array}{ccc}\mathcal{D}_2 & & \mathcal{D}_n \\ \vdots & & \vdots \\ \Gamma \Rightarrow \Pi & \ldots & \Gamma \Rightarrow \Pi\end{array}}{\Gamma \Rightarrow \Pi}\,(B_{\mathcal{S}''})$$

in such a way that:

- if a top rule in $\mathcal{D}_1$ belonging to $\mathcal{S}$ occurs above a top rule of one among $\mathcal{S}_1, \ldots, \mathcal{S}_n$, then its copy in $\mathcal{D}'_1$ belongs to $\mathcal{S}'$,
- if a top rule in $\mathcal{D}_1$ belonging to $\mathcal{S}$ occurs below a top rule of one among $\mathcal{S}_1, \ldots, \mathcal{S}_n$, then its copy in $\mathcal{D}'_1$ belongs to $\mathcal{S}''$.

Notice that in the obtained derivation no top rule of $\mathcal{S}'$ occurs below any top rule of $\mathcal{S}_1, \ldots, \mathcal{S}_n$, and no top rule of $\mathcal{S}''$ occurs above any top rule of $\mathcal{S}_1, \ldots, \mathcal{S}_n$. Moreover, also due to Lemma 3.2.2:

($*$) neither $\mathcal{S}'$ and $\mathcal{S}''$ nor two copies of the same 2-system instance in $\mathcal{D}_2, \ldots, \mathcal{D}_n$ can be entangled or have top rules along the same path of the derivation.

The idea of a strategy to apply e-reductions that leads to the required derivation $\mathcal{P}'$ is the following. We start reducing one of the 2-system instances with lowermost bottom rule. Whenever we apply an e-reduction we collect all entangled copies of the same 2-system instance in the same class of instances. We continue the disentanglement focusing on a single class and reducing all its elements before we move on to another class. Notice that the number of classes never increases and is bounded by the number of 2-system instances in the original derivation. If we, moreover, fix a class and regard as more complex the 2-system instances which are entangled with more different instances, then the strategy guarantees that the elements of the class are disentangled one by one without duplicating other maximally complex elements of the same class.

To formalize this strategy, we introduce some auxiliary notions. We define the equivalence relation $\sim$ as the transitive and symmetric closure of the binary relation that holds between a 2-system instance and any of its copies generated by an e-reduction – notice that e-reductions do not only copy $\mathcal{S}$ but also the 2-system instances in $\mathcal{D}_2, \ldots, \mathcal{D}_n$. Given any 2-system derivation $\mathcal{P}$, let us denote by $E^{\mathcal{P}}$ the set of all entangled 2-system instances in $\mathcal{P}$, and by $E^{\mathcal{P}}/_{\sim}$ the quotient set of $E^{\mathcal{P}}$ w.r.t. the equivalence relation $\sim$. Moreover, we denote by $\mathcal{S}^{low}$ the 2-system instance in $E^{\mathcal{P}}$ which has the lowest and leftmost bottom rule in $\mathcal{P}$. Finally, we compute the *entanglement number* (*e-number* for short) of a 2-system instance $\mathcal{S}$ as follows: for each derivation $\mathcal{D}$ of a premise of the bottom rule of $\mathcal{S}$ we count the number of equivalence classes containing 2-system instances that have top rules in $\mathcal{D}$ and are entangled with $\mathcal{S}$, then we sum all the resulting numbers up to obtain the e-number of $\mathcal{S}$.

We prove now the statement of the lemma by induction on the lexicographical ordered of the triples (κ, μ, ν) where, fixed a derivation $\mathcal{P}$,

- κ is the cardinality of $E^{\mathcal{P}}/_{\sim}$, i.e. the number of classes of entangled 2-system instances,
- μ is the maximum e-number of the elements of $[\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}}/_{\sim}$,
- ν is the number of elements of $[\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}}/_{\sim}$ with e-number μ.

Base case. If either κ, μ or ν are equal to 0, then no 2-system instance is entangled. Otherwise, first, $E^{\mathcal{P}}/_{\sim}$ would contain at least one element, and $e \geq 1$. Second, $[\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}}/_{\sim}$ would not be empty and both μ and ν would be greater than 0.

Inductive step. Given any 2-system derivation $\mathcal{P}$ with complexity $(\kappa, \mu, \nu) \geq (1, 1, 1)$ we transform it into a 2-system derivation $\mathcal{P}'$ with complexity smaller than (κ, μ, ν). We obtain $\mathcal{P}'$ applying an arbitrary e-reduction to an uppermost element $\mathcal{S} \in [\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}}/_{\sim}$ with e-number μ.

First notice that we never increase κ. Moreover, if $\nu > 1$ we reduce ν without increasing μ and if $\nu = 1$ and $\mu > 1$ we reduce μ. Indeed, after the e-reduction all top rules of $\mathcal{S}$ that were involved in an entanglement with the elements of some class $[\mathcal{S}']_{\sim} \in E^{\mathcal{P}}/_{\sim}$ above the same premise of $(B_{\mathcal{S}})$, are no more involved in such entanglement. This holds because, due to $(*)$ and the definition of $\sim$, the top rules of elements contained in $[\mathcal{S}']_{\sim} \in E^{\mathcal{P}}/_{\sim}$ cannot occur along the same path of the derivation. In general we never increase neither μ nor ν, because if we duplicate a 2-system instance during an e-reduction, either it did not belong to $[\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}}/_{\sim}$ and hence the copies do not belong to $[\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}'}/_{\sim}$, or it did not have maximal entanglement number w.r.t. the class $[\mathcal{S}^{low}]_{\sim} \in E^{\mathcal{P}}/_{\sim}$, because we always e-reduce a topmost 2-system instance among those with maximal e-number in $[\mathcal{S}^{low}]_{\sim}$. Finally, we change the considered class $[\mathcal{S}^{low}]_{\sim}$ only when it is empty, because our e-reduction strategy chooses $\mathcal{S}^{low}$ only if $[\mathcal{S}^{low}]_{\sim}$ is a singleton. If $\nu = 1$ and $\mu = 1$ we reduce κ. Indeed, we replace the unique element of $[\mathcal{S}^{low}]_{\sim}$ with non-entangled 2-system instances and $[\mathcal{S}^{low}]_{\sim}$ does not belong to $E^{\mathcal{P}'}/_{\sim}$. □

3.2.2 From hypersequent to 2-system derivations

Given any set $\mathbb{HR}$ of hypersequent rules and set $\mathbb{SR}$ of 2-systems s.t. if $Hr \in \mathbb{HR}$ then $Sys_{Hr} \in \mathbb{SR}$. Starting from a derivation in $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ we construct a derivation in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ of the same end-sequent.

The algorithm

Input: a derivation $\mathcal{D}$ of a sequent $\Gamma \Rightarrow \Pi$ in $\mathrm{H}\mathfrak{I} + \mathbb{HR}$. Output: a derivation $\mathcal{D}'$ of $\Gamma \Rightarrow \Pi$ in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$.

Intuitively, if an $\mathrm{H}\mathfrak{I}$ rule corresponds to an $\mathrm{L}\mathfrak{I}$ rule, then we translate the applications of the former in $\mathcal{D}$ with applications of the latter in $\mathcal{D}'$.

We cannot directly translate the applications of the $\mathrm{H}\mathfrak{I}$ rules (EW) and (EC) because they do not correspond to any $\mathrm{L}\mathfrak{I}$ rule. We avoid to translate them by only considering derivations $\mathcal{D}$ in which (i) all applications of (EC) occur immediately above the root, and (ii) all applications of (EW) occur where immediately needed, namely where they introduce context components required by other rule applications. As we will show each hypersequent derivation of a sequent can be transformed into an equivalent one of this form.

Since the rules in $\mathbb{HR}$ are non-local, we need to translate them in two steps. Thus, we consider the uppermost application of (EC) in $\mathcal{D}$ and we construct a *partial derivation* for each component of its premise. Such a derivation contains applications of $\mathrm{L}\mathfrak{I}$ rules and of the top rules of the 2-systems in $\mathbb{SR}$ without any applicability condition, see Lemma 3.2.4. The desired derivation $\mathcal{D}'$ is then obtained by suitably applying to these partial derivations the bottom rules corresponding to the employed top rules (Theorem 3.2.5).

Definition 3.2.2. A partial derivation in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ is a derivation in $\mathrm{L}\mathfrak{I}$ extended with the top rules of $\mathbb{SR}$ (without their applicability conditions relative to a bottom rule application).

We show an example of the first part of the translation to guide the reader's intuition through the proofs that follow.

Example 3.2.3. Consider the $\mathrm{H}\mathfrak{I} + (\mathrm{com})$ derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \Rightarrow C}{A, C \Rightarrow C} \quad B \Rightarrow B}{A, B \Rightarrow C \mid C \Rightarrow B}\,(\mathrm{com})' \quad \dfrac{\dfrac{C \Rightarrow C}{C, B \Rightarrow C} \quad A \Rightarrow A}{A, B \Rightarrow C \mid C \Rightarrow A}\,(\mathrm{com})''}{A, B \Rightarrow C \mid C \Rightarrow B \wedge A}\,(\wedge r)}{A \wedge B \Rightarrow C \mid C \Rightarrow B \wedge A}}{A \wedge B \Rightarrow C \mid\Rightarrow C \to B \wedge A}}{\Rightarrow A \wedge B \to C \mid\Rightarrow C \to B \wedge A}}{\Rightarrow A \wedge B \to C \mid\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)}}{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A) \mid\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)}\,(\mathrm{EC})}{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)}$$

and observe that it satisfies property (i) and, trivially, property (ii). The partial derivations in $\mathrm{L}\mathfrak{I} + \mathit{Sys}_{(\mathrm{com})}$ (see Example 3.1.2) of the components of the uppermost application of (EC) in the above proof are:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \Rightarrow C}{A, C \Rightarrow C}}{A, B \Rightarrow C}\,(\mathrm{com}_1)' \quad \dfrac{\dfrac{C \Rightarrow C}{C, B \Rightarrow C}}{A, B \Rightarrow C}\,(\mathrm{com}_1)''}{A, B \Rightarrow C}\,\mathit{dummy}}{A \wedge B \Rightarrow C}}{\Rightarrow A \wedge B \to C}}{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{A \Rightarrow A}{C \Rightarrow A}\,(\mathrm{com}_2)'' \quad \dfrac{B \Rightarrow B}{C \Rightarrow B}\,(\mathrm{com}_2)'}{C \Rightarrow B \wedge A}\,(\wedge r)}{\Rightarrow C \to B \wedge A}}{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)}$$

where $(\mathrm{com}_1)'$ and $(\mathrm{com}_2)'$ translate $(\mathrm{com})'$ while $(\mathrm{com}_1)''$ and $(\mathrm{com}_2)''$ translate $(\mathrm{com})''$. Notice that in order to handle the context component duplication relative to $(\wedge r)$, we apply a *dummy* bottom rule.

The partial derivations obtained have the same structure as the hypersequent derivations of the corresponding components (see *ancestor tree* in Definition 3.2.6).

We use the following two definitions to formalize the properties of derivations needed to handle (EW) and (EC).

Definition 3.2.3. For any one-premise rule (r) we call a *queue of* (r) any sequence of consecutive applications of (r) that is neither immediately preceded nor immediately followed by applications of (r).

Definition 3.2.4. We say that an $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ derivation is in *structured form iff* all (EC) applications occur in a queue immediately above the root, and all (EW) applications occur in subderivations of the form

$$\dfrac{\dfrac{\dfrac{G_1 \mid S_1}{\vdots}\,(\mathrm{EW})}{G \mid S_1}\,(\mathrm{EW}) \quad \ldots \quad \dfrac{\dfrac{G_n \mid S_n}{\vdots}\,(\mathrm{EW})}{G \mid S_n}\,(\mathrm{EW})}{G \mid S_0}\,(r)$$

where (r) is any rule with more than one premise and each component of G is contained in at least one of the hypersequents $G_1, \ldots, G_n$.

A derivation in structured form can be divided into a part containing only (EC) applications and a part containing the applications of any other rule. We introduce notation for the hypersequent separating the two parts.

Definition 3.2.5. If $\mathcal{D}$ is a derivation in structured form, we denote by $\widehat{H}_{\mathcal{D}}$ the premise of the uppermost application of (EC) in $\mathcal{D}$.

Definition 3.2.6. Given a $\mathrm{H}\mathfrak{I}+\mathbb{HR}$ derivation. A hypersequent component S' is a *parent* of a hypersequent component S, denoted as $p(S, S')$, if one of the following conditions holds:

- S is active in the conclusion of an application of some $Hr \in \mathbb{HR}$, and S' is the active component of a premise linked to S, see Definition 3.1.1;
- S is active in the conclusion of an application of a rule of $\mathrm{H}\mathfrak{I}$, and S' is the active component of a premise of such application;
- S is a context component in the conclusion of any rule application, and S' is the corresponding context component in a premise of such application.

We say that a hypersequent component S' is an *ancestor* of a hypersequent component S, and we write $a(S, S')$, if the pair (S, S') is in the transitive closure of the relation $p(\ ,\ )$. The *ancestor tree* of a hypersequent component S is the tree whose nodes are all hypersequent components related to S by $a(\ ,\ )$ and whose edges are defined by the relation $p(\ ,\ )$ between such nodes.

We prove below that from any $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ derivation $\mathcal{D}$ of a sequent we can construct a partial derivation for each component of $\widehat{H}_{\mathcal{D}}$ having the *same structure* as the ancestor tree of that component, namely consisting of the translation of the rules in the ancestor tree, with the exception of (EW).

Remark 3.1.

- In an $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ derivation that does not use (EC), the ancestor tree of each hypersequent is a sequent derivation.
- If S is the active component of an application of (EW), then there is no S' such that $p(S, S')$.

Lemma 3.2.4. *Let $\mathbb{HR}$ be a set of hypersequent rules and $\mathbb{SR}$ of 2-systems s.t. if $Hr \in \mathbb{HR}$ then $Sys_{Hr} \in \mathbb{SR}$. Given any $\mathrm{H}\mathfrak{I}+\mathbb{HR}$ derivation $\mathcal{D}$ in structured form, for each component S of $\widehat{H}_{\mathcal{D}}$ we can construct a partial derivation in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ having the same structure as the ancestor tree of S in $\mathcal{D}$.*

Proof. Let H be a hypersequent in $\mathcal{D}$ derived without using (EC). We construct a partial derivation in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ with the required property for each of its components. The proof proceeds by induction on the height l of the derivation of H by translating each rule of $\mathrm{H}\mathfrak{I} + \mathbb{HR}$, with the exception of (EW), into the corresponding sequent rule in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$.

Base case. If $l = 1$, which means that H is an axiom, then the partial derivation in $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ simply contains H.

Inductive step. We consider the last rule $(r) \neq$ (EW) applied in the subderivation $\mathcal{D}'$ of H, and we distinguish the two cases: (i) (r) is a one-premise rule and (ii) (r) has more premises; for the latter case, since $\mathcal{D}'$ is in structured form, we deal also with possible queues of (EW) above its premises.

1. Assume that the derivation ending in a one-premise rule $(r) \in \mathrm{H}\mathfrak{I}$ is

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \vdots\\ G \mid S\end{array}}{G \mid S'}\ (r)$$

   By induction hypothesis there is a partial derivation of S having the same structure as the ancestor tree of S. The partial derivation of S' is simply obtained by applying (r).

   The case in which (r) is a one-premise rule belonging to $\mathbb{HR}$ is a special case of (ii) for which there is no need to consider queues of (EW).

2. Assume that $(r) = (Hr) \in \mathbb{HR}$ and that the derivation $\mathcal{D}'$, of height n, is the following

$$\dfrac{\begin{array}{cccccccc} \mathcal{D}_1^1 & & \mathcal{D}_1^{m_1} & & \mathcal{D}_k^1 & & \mathcal{D}_k^{m_k} \\ \vdots & & \vdots & & \vdots & & \vdots \\ G \mid S_1^1 & \dots & G \mid S_1^{m_1} & \dots & G \mid S_k^1 & \dots & G \mid S_k^{m_k} \end{array}}{G \mid \Delta_1, \Gamma_1 \Rightarrow \Pi_1 \mid \dots \mid \Delta_k, \Gamma_k \Rightarrow \Pi_k}\ (Hr)$$

where the premises $G \mid S_j^i$ of (Hr) are possibly inferred by a queue of (EW). When this is the case, we consider the uppermost hypersequents in the queues. More precisely, we consider the following derivations, each of which has height strictly less than n:

$$\begin{array}{cccccccc} \mathcal{D}_1^1 & & \mathcal{D}_1^{m_1} & & \mathcal{D}_k^1 & & \mathcal{D}_k^{m_k} \\ \vdots & & \vdots & & \vdots & & \vdots \\ G_1^1 \mid S_1^1 & \dots & G_1^{m_1} \mid S_1^{m_1} & \dots & G_k^1 \mid S_k^1 & \dots & G_k^{m_k} \mid S_k^{m_k} \end{array}$$

where, for $1 \leq y \leq k$ and $1 \leq x \leq m_y$, the hypersequent G_y^x is G if there is no (EW) application immediately above $G \mid S_y^x$; otherwise, $G_y^x \mid S_y^x$ is the premise of the uppermost (EW) application in the queue immediately above $G \mid S_y^x$.

Since $\mathcal{D}$, and hence $\mathcal{D}'$, is in structured form, each component of G must occur in at least one of the hypersequents $G_1^1, \dots, G_1^{m_1}, \dots, G_k^1, \dots, G_k^{m_k}$. We obtain partial derivations for $\Delta_1, \Gamma_1 \Rightarrow \Pi_1, \dots, \Delta_k, \Gamma_k \Rightarrow \Pi_k$ applying the top rules of the 2-system Sys_{Hr} as follows

$$\dfrac{S_1^1 \quad \dots \quad S_1^{m_1}}{\Delta_1, \Gamma_1 \Rightarrow \Pi_1}\ (r_1) \qquad \dots \qquad \dfrac{S_k^1 \quad \dots \quad S_k^{m_k}}{\Delta_k, \Gamma_k \Rightarrow \Pi_k}\ (r_k)$$

Indeed, by induction hypothesis, we have a partial derivation for each S_y^x. In case a component S of G occurs in more than one premise, we have different partial derivations and we apply a dummy bottom rule

$$\dfrac{S \quad \dots \quad S}{S}$$

to obtain one partial derivation from them.

The obtained partial derivations clearly satisfy the following property: with the exception of (EW) and of dummy bottom rules, a rule application occurs in the ancestor tree of a hypersequent component in $\mathcal{D}$ *iff* its translation occurs in the partial derivation of such component. □

The next step of the translation consists in applying a bottom rule for each group of top rules translating one hypersequent rule application. If we applied dummy bottom rules inside the partial derivations, we might be forced to apply a single bottom rule for

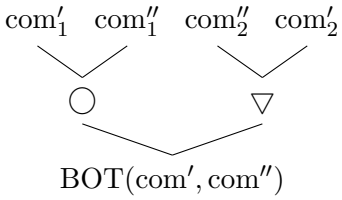
more than one of such groups – thus creating what will be called a *mixed system*. In Theorem 3.2.5 we prove that we can always restructure the derivation and obtain the desired exact match between groups of top rules and bottom rules. We first show an example that clarifies the main ideas exploited in the following proof.

Example 3.2.4. Consider the partial derivations obtained in Example 3.2.3, if we apply a bottom rule to them we obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \Rightarrow C}{A, C \Rightarrow C}}{A, B \Rightarrow C}\,(\mathrm{com}_1)' \quad \dfrac{\dfrac{C \Rightarrow C}{C, B \Rightarrow C}}{A, B \Rightarrow C}\,(\mathrm{com}_1)''}{A, B \Rightarrow C}\,\textit{dummy}}{A \wedge B \Rightarrow C}}{\Rightarrow A \wedge B \to C} \qquad \dfrac{\dfrac{\dfrac{A \Rightarrow A}{C \Rightarrow A}\,(\mathrm{com}_2)'' \quad \dfrac{B \Rightarrow B}{C \Rightarrow B}\,(\mathrm{com}_2)'}{C \Rightarrow B \wedge A}\,(\to r)}{\Rightarrow C \to B \wedge A}}{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)} \qquad \dfrac{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A) \quad \Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)}{\Rightarrow (A \wedge B \to C) \vee (C \to B \wedge A)}\,(\mathrm{com}_{bot})$$

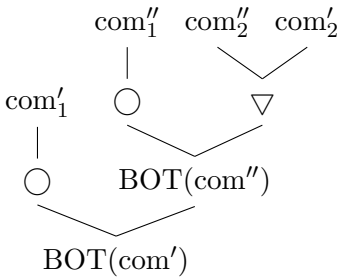
where (com_{bot}) is the bottom rule both for $(\mathrm{com}_1)'$ and $(\mathrm{com}_2)'$ and for $(\mathrm{com}_1)''$ and $(\mathrm{com}_2)''$. We call this a mixed system.

We can abstract this derivation as



where we represent by $\mathrm{BOT}(\mathrm{com}', \mathrm{com}'')$ the bottom rule of com' and com'', by $\bigcirc$ the forks in the derivation tree corresponding to dummy bottom rules, and by $\bigtriangledown$ the forks corresponding to non-dummy rules.

Given that the removal of premises from the $\bigcirc$ forks is a logically sound operation, we transform the structure of the derivation as follows:



Now the group of top rules translating com′ and the one translating com″ have different bottom rules. The derivation resulting from this is the following

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \Rightarrow C}{A, C \Rightarrow C}}{A, B \Rightarrow C}\,(\mathrm{com}_1)'}{A \wedge B \Rightarrow C}}{\Rightarrow A \wedge B \to C}}{\Rightarrow \alpha} \qquad \dfrac{\dfrac{\dfrac{\dfrac{C \Rightarrow C}{C, B \Rightarrow C}}{A, B \Rightarrow C}\,(\mathrm{com}_1)''}{A \wedge B \Rightarrow C}}{\Rightarrow A \wedge B \to C}}{\Rightarrow \alpha} \qquad \dfrac{\dfrac{\dfrac{A \Rightarrow A}{C \Rightarrow A}\,(\mathrm{com}_2)'' \quad \dfrac{B \Rightarrow B}{C \Rightarrow B}\,(\mathrm{com}_2)'}{C \Rightarrow B \wedge A}}{\Rightarrow C \to B \wedge A}}{\Rightarrow \alpha}\,(\mathrm{com}_{bot})''$$

$$\dfrac{\Rightarrow \alpha \qquad \Rightarrow \alpha}{\Rightarrow \alpha}\,(\mathrm{com}_{bot})'$$

where α is the formula $(A \wedge B \to C) \vee (C \to B \wedge A)$.

Theorem 3.2.5. *For any set $\mathbb{HR}$ of hypersequent rules and set $\mathbb{SR}$ of 2-systems s.t. if $Hr \in \mathbb{HR}$ then $Sys_{Hr} \in \mathbb{SR}$, if $\vdash_{\mathrm{H}\mathfrak{I}+\mathbb{HR}} \Gamma \Rightarrow \Pi$ then $\vdash_{\mathrm{L}\mathfrak{I}+\mathbb{SR}} \Gamma \Rightarrow \Pi$ and if $\vdash^{c.f.}_{\mathrm{H}\mathfrak{I}+\mathbb{HR}} \Gamma \Rightarrow \Pi$ then $\vdash^{c.f.}_{\mathrm{L}\mathfrak{I}+\mathbb{SR}} \Gamma \Rightarrow \Pi$.*

Proof. Let $\mathcal{D}$ be a $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ derivation of $\Gamma \Rightarrow \Pi$. By the results in Section 3.2.2 we can assume that $\mathcal{D}$ is in structured form. By applying the procedure of Lemma 3.2.4 to the premise $\widehat{H}_{\mathcal{D}}$ of the uppermost application of (EC) in $\mathcal{D}$ we obtain a set of partial derivations $\{\mathcal{D}_i\}_{i \in I}$ whose rules translate those occurring in the ancestor trees of each component of $\widehat{H}_{\mathcal{D}}$.

We show that we can suitably apply the bottom rules of 2-systems in $\mathbb{SR}$ to the roots of $\{\mathcal{D}_i\}_{i \in I}$ in order to obtain the required $\mathrm{L}\mathfrak{I} + \mathbb{SR}$ derivation of $\Gamma \Rightarrow \Pi$. First, we group all top rule applications in $\{\mathcal{D}_i\}_{i \in I}$ according to the application of $Hr \in \mathbb{HR}$ that these rules translate. For each such group we apply one bottom rule below the partial derivations in which the top rules of the group occur. As shown in Example 3.2.4, due to the duplication of context sequents in hypersequent rules which we handle using dummy bottom rules, we may need to apply a single bottom rule below groups of top rules translating different hypersequent rules. In particular, if we consider the algorithm, it is easy to see that this can only happen when

(*i*) two hypersequent rule applications (h') and (h'') occur above different premises of a rule application (r);

(*ii*) S_0 is an active component and $S_1, \ldots, S_n$ are context components occurring in the conclusion of (r);

(*iii*) both (h') and (h'') have active components including different ancestors of some S_i for $0 \leq i \leq n$.

Indeed, if (h') and (h'') occur one above the other, since each premise of a hypersequent rule contains one active component, the two groups of partial derivations containing the top rules translating (h') and (h'') share only one partial derivation, and hence no

mixed system is produced because we can just apply one bottom rule below the other. Moreover, if one between (h') and (h'') have no active component which is an ancestor of some S_i, then the two groups of partial derivations containing the top rules translating (h') and (h'') share no partial derivation and no mixed system is produced.

If (i)–(iii) hold, the top rules translating (h') and (h'') occur above different premises of a non-dummy rule with conclusion S_0 – just like the two applications of (com_2) in Example 3.2.4 – and of some dummy bottom rules with conclusions $S_1, \ldots, S_n$ – just like the two applications of (com_1) in Example 3.2.4. When we apply a bottom rule for such a group of top rules we obtain a *mixed system*, namely a 2-system that contains top rules translating different hypersequent rule applications.

We show that we can replace each mixed system by regular 2-systems. First notice that the following facts are always true:

1. if we remove all premises but one from a dummy bottom rule in a partial derivation we still obtain a partial derivation of the same sequent;

2. every time a pair of top rules translating different hypersequent rule applications occur in the same mixed system above different premises of a non-dummy rule, all other pairs of top rules translating these two hypersequent rule applications occur above different premises of dummy bottom rules.

While the truth of 1 is simply due to the shape of dummy bottom rules, that of 2 is evident if we carefully consider the hypersequent derivation configurations that lead to the creation of a mixed system. Indeed, as for 2, the partial derivation translating the ancestor tree of S_0 mentioned in (i) contains the non-dummy rule mentioned in 2, and the translations of the ancestor trees of $S_1, \ldots, S_n$ contain the dummy bottom rules mentioned in 2. Moreover, such non-dummy rules cannot be more than one, since every premise of (r) contains only one active component.

From 1 it follows that if two top rules occur above different premises of a dummy bottom rule, we can remove one of them from the partial derivation containing the other. If we do so, we say that we *split* the dummy bottom rule.

Consider now a mixed system

$$\dfrac{\begin{array}{ccc}\mathcal{D}_1 & & \mathcal{D}_k \\ \vdots & & \vdots \\ \Gamma \Rightarrow \Delta & \ldots & \Gamma \Rightarrow \Delta\end{array}}{\Gamma \Rightarrow \Delta}$$

where the derivation $\mathcal{D}_i$, for $1 \leq i \leq k$, contains the rule applications $(r_i^1), \ldots, (r_i^n)$. We adopt the convention that the rules with same superscript index translate the same hypersequent rule.

To replace such mixed system with regular 2-systems we proceed as follows. First we replace the mixed system with a 2-system for the group of top rules with superscript 1:

$$\dfrac{\begin{array}{ccc}\mathcal{D}'_1 & & \mathcal{D}'_k\\ \vdots & & \vdots\\ \Gamma \Rightarrow \Delta & \dots & \Gamma \Rightarrow \Delta\end{array}}{\Gamma \Rightarrow \Delta}\ (b^1)$$

where $\mathcal{D}'_1, \dots, \mathcal{D}'_k$ only contain the rules $(r^1_1), \dots, (r^1_k)$ and those top rules that cannot be removed from the partial derivations by splitting dummy bottom rules – if we need to choose, we pick the top rules with minimum superscript index. After this, we introduce further bottom rules as follows

$$\dfrac{\dfrac{\begin{array}{ccccc}\mathcal{D}'_1 & \mathcal{D}''_2 & & \mathcal{D}''_k & \\ \vdots & \vdots & & \vdots & \\ \Gamma \Rightarrow \Delta & \Gamma \Rightarrow \Delta & \dots & \Gamma \Rightarrow \Delta & \end{array}}{\Gamma \Rightarrow \Delta}\ (b^2) \quad \dots \quad \dfrac{\begin{array}{ccccc} \mathcal{D}''_1 & & \mathcal{D}''_{k-1} & & \mathcal{D}'_k\\ \vdots & & \vdots & & \vdots\\ \Gamma \Rightarrow \Delta & \dots & \Gamma \Rightarrow \Delta & \Gamma \Rightarrow \Delta & \end{array}}{\Gamma \Rightarrow \Delta}\ (b^2)}{\Gamma \Rightarrow \Delta}\ (b^1)$$

where the bottom rules (b^2) are only introduced below the branches $\mathcal{D}'_1, \dots, \mathcal{D}'_k$ containing some of the rules $(r^2_1), \dots, (r^2_k)$, and the derivations $\mathcal{D}''_1, \dots, \mathcal{D}''_k$ are copies of $\mathcal{D}_1, \dots, \mathcal{D}_k$ only containing $(r^2_1), \dots, (r^2_k)$ and those top rules that cannot be removed by splitting dummy bottom rules. We keep duplicating the derivation in such way until either we do not need any more bottom rules or we introduced bottom rules for all superscript indices $1, \dots, n$. Given that we can add bottom rules for all groups of top rules in the mixed system, in order to be sure that the result does not contain any mixed system we only need to show that we never add a top rule application above the wrong premise of its bottom rule. For the sake of contradiction suppose that we do. We add a top rule application (r^i_p) above a wrong premise of its bottom rule only if we just introduced a new bottom rule (b^j), for $i < j \leq n$, and we cannot remove (r^i_p) – by splitting a dummy bottom rule – from the derivation containing a top rule (r^j_p) that we need in the branch that we are considering. But if we cannot remove (r^i_p) from the partial derivation containing (r^j_p), by 2 we can remove any (r^j_q) from any partial derivation containing any (r^i_q), as long as $q \neq p$. Given that the bottom rule (b^i) occurs below (b^j), it follows that there is no top rule (r^j_q) above the considered premise of the bottom rule (b^i). Hence we can infer that (r^j_p) is not needed and we do not need to add (r^i_p) in the first place, contrarily to the assumptions.

Notice that the procedure does not require all groups of top rules to have exactly k elements. If, for example, the group with superscript index i contains l top rule applications for $l < k$, then the bottom rules for i will have l premises. This does not influence any other group of top rules.

Thus, we eventually obtain an $\mathrm{L}\mathfrak{J} + \mathbb{SR}$ derivation of $\Gamma \to \Pi$. □

Normal forms of hypersequent derivations

In the previous algorithm we only considered hypersequent derivations in structured form, namely in which (EC) applications occur immediately above the root and (EW) applications occur where needed. Here we show how to transform each hypersequent derivation into a derivation in structured form.

Definition 3.2.7. The *external contraction rank (ec-rank)* of an application E of (EC) in a derivation is the number of applications of rules other than (EC) between E and the root of the derivation.

Lemma 3.2.6. *Each* $\mathrm{H}\mathfrak{I} + \mathbb{HR}$ *derivation* $\mathcal{D}$ *can be transformed into a derivation of the same end-hypersequent in which all* (EC) *applications have ec-rank* 0*.*

Proof. Proceed by double induction on the lexicographically ordered pair (μ, ν), where μ is the maximum ec-rank of any (EC) application in $\mathcal{D}$, and ν is the number of (EC) applications in $\mathcal{D}$ with maximum ec-rank.

Base case. If $\mu = 0$ the claim trivially holds.

Inductive step. Assume that $\mathcal{D}$ has maximum ec-rank μ and that there are ν applications of the rule (EC) with ec-rank μ. We show how to transform $\mathcal{D}$ into a derivation $\mathcal{D}'$ having either maximum ec-rank $\mu' < \mu$ or ec-rank μ and number of (EC) applications with maximum ec-rank $\nu' < \nu$.

Consider an (EC) application with ec-rank μ in $\mathcal{D}$ and the queue of (EC) containing it. There cannot be any applications of (EC) above this queue because the ec-rank of its elements is maximal. We distinguish cases according to the rule (r) applied to the conclusion of the last element of such queue.

Assume that (r) has one premise. If $(r) = (\mathrm{EW})$, we apply (EW) – with the same active component – before the queue. If $(r) \neq (\mathrm{EW})$, we apply (r) immediately before the queue, possibly followed by applications of (EC).

Notation. Given a hypersequent H we denote by $(H)^u$ the hypersequent $H \mid \ldots \mid H$ containing u copies of H $(u \geq 0)$.

Let (r) be any external context-sharing rule with more than one premise and consider any subderivation of $\mathcal{D}$ of the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\vdots}}{G \mid G'_1 \mid (C_1)^{m_1}}\,(\mathrm{EC})}{\vdots}\,(\mathrm{EC})}{G \mid C_1} \quad \ldots \quad \dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_n}{\vdots}}{G \mid G'_n \mid (C_n)^{m_n}}\,(\mathrm{EC})}{\vdots}\,(\mathrm{EC})}{G \mid C_n}\,(r)}{G \mid H}$$

where G'_i, for $1 \leq i \leq n$, only contains components in G and the derivations $\mathcal{D}_1, \ldots, \mathcal{D}_n$ contain no application of (EC). We can transform $\mathcal{D}$ into a derivation $\mathcal{D}'$ in which all applications of (EC) occurring above the hypersequent $G \mid H$ are either immediately above it or immediately above another application of (EC); their ec-rank is reduced by 1 because (r) does not occur below them anymore.

We first prove that $(*)$ the hypersequent $G \mid G'' \mid (H)^q$, where $G'' = G'_1 \mid \ldots \mid G'_n$ and $q = (\sum_{i=1}^{n}(m_i - 1)) + 1$ is derivable from

$$G \mid G'_1 \mid (S_1)^{m_1} \ , \ \ldots \ , \ G \mid G'_n \mid (S_n)^{m_n}$$

using only (EW) and (r). The hypersequent $G \mid H$ then follows from $G \mid G'' \mid (H)^q$ by (EC) as all the components of G'' occur also in G. The obtained derivation $\mathcal{D}'$ has maximum ec-rank $\mu' < \mu$, or the occurrences of (EC) with ec-rank μ occurring in it are $\nu' < \nu$.

It remains to prove claim $(*)$. We have a derivation of any element of the set

$$\mathbb{Q} = \{G \mid G'' \mid (H)^0 \mid (S_1)^{x_1} \mid \ldots \mid (S_n)^{x_n} \ : \ \sum_{i=1}^{n} x_i \ = \ (\sum_{i=1}^{n}(m_i - 1)) + 1\}$$

from the hypersequents $G \mid G'_1 \mid (S_1)^{m_1} \ , \ \ldots \ , \ G \mid G'_n \mid (S_n)^{m_n}$ using only (EW). Indeed for any hypersequent in $\mathbb{Q}$ and for $1 \leq i \leq n$, there is at least one $x_i \geq m_i$, because otherwise $\sum_{i=1}^{n} x_i \ < \ (\sum_{i=1}^{n}(m_i - 1)) + 1$. The claim $(*)$ therefore follows by Lemma 3.2.7 below being $G \mid G'' \mid (H)^q$ the only element of the set

$$\mathbb{Q}' = \{G \mid G'' \mid (H)^q \mid (S_1)^{x_1} \mid \ldots \mid (S_n)^{x_n} \ : \ \sum_{i=1}^{n} x_i \ = \ 0\}$$

for $q = (\sum_{i=1}^{n}(m_i - 1)) + 1$. □

The following is the central lemma of the previous proof.

Lemma 3.2.7. *For any application of a hypersequent rule*

$$\frac{G \mid S_1 \qquad \ldots \qquad G \mid S_n}{G \mid H} \ (r)$$

and natural number $d \geq 0$, consider the set of hypersequents

$$\mathbb{L}_d = \{G \mid (H)^c \mid (S_1)^{x_1} \mid \ldots \mid (S_n)^{x_n} \ : \ \sum_{i=1}^{n} x_i = d\}$$

where G, H are hypersequents, $S_1, \ldots, S_n$ sequents, and c is a natural number. For any natural number e, s.t. $0 \leq e \leq d$, each element of the set

$$\mathbb{L}_{(d-e)} = \{G \mid (H)^{c+e} \mid (S_1)^{x'_1} \mid \ldots \mid (S_n)^{x'_n} \ : \ \sum_{i=1}^{n} x'_i = d - e\}$$

is derivable from hypersequents in $\mathbb{L}_d$ by repeatedly applying the rule (r).

Proof. By induction on e.

Base case. If $e = 0$, then $\mathbb{L}_d = \mathbb{L}_{d-e}$.

Inductive step. Assume that $e > 0$ and that the claim holds for all $e' < e$. By induction hypothesis there exists a derivation from the hypersequents in $\mathbb{L}_d$ for each element of the set

$$\mathbb{L}_{(d-(e-1))} = \{G \mid (H)^{c+(e-1)} \mid (S_1)^{x''_1} \mid \ldots \mid (S_n)^{x''_n} \ : \ \sum_{i=1}^{n} x''_i = d - (e-1)\}$$

that only consists of applications of (r). Any hypersequent

$$G \mid (H)^{c+e} \mid (S_1)^{x'_1} \mid \ldots \mid (S_n)^{x'_n}$$

in $\mathbb{L}_{(d-e)}$ can be derived from elements of $\mathbb{L}_{(d-(e-1))}$ as follows:

$$\frac{G \mid (H)^{c+(e-1)} \mid H'_1 \quad \ldots \quad G \mid (H)^{c+(e-1)} \mid H'_n}{G \mid (H)^{c+e} \mid (S_1)^{x'_1} \mid \ldots \mid (S_n)^{x'_n}} \ (r)$$

where, for $1 \leq i \leq n$, $H'_i = (S_1)^{y_1} \mid \ldots \mid (S_n)^{y_n}$ is such that if $j \neq i$ then $y_j = x'_j$ and if $j = i$ then $x'_j + 1$; i.e., the components $S_1, \ldots, S_n \notin G$ occur in the i^{th} premise as many times as in the conclusion, except for S_i which occurs one more time.

All premises of this rule application are hypersequents in $\mathbb{L}_{(d-(e-1))}$, indeed

$$(x'_1 + 1) + x'_2 + \ldots + x'_n \ = \ \ldots \ = \ x'_1 + \ldots + x'_{n-1} + (x'_n + 1) \ = \ (\sum_{i=1}^{n} x'_i) + 1$$

and

$$(\sum_{i=1}^{n} x'_i) + 1 \ = \ (d - e) + 1 \ = \ d - (e - 1)$$

Given that only the rule (r) is used to derive the elements of $\mathbb{L}_{d-(e-1)}$ from the elements of $\mathbb{L}_d$, also the elements of $\mathbb{L}_{(d-e)}$ can be derived from those of $\mathbb{L}_d$ by applying only (r). □

Lemma 3.2.8. *Any* $\mathrm{H}\mathfrak{I}+\mathbb{HR}$ *derivation of a sequent can be transformed into a derivation in structured form.*

Proof. Let $\mathcal{D}$ be a hypersequent derivation of a sequent S in $\mathrm{H}\mathfrak{I} + \mathbb{HR}$. By Lemma 3.2.6 we can assume that all applications of (EC) in $\mathcal{D}$ occur in a queue immediately above S. Consider an application of (EW), with premise G and conclusion $G \mid S$, which is not as in Definition 3.2.4. First notice that $G \mid S$ cannot be the root of $\mathcal{D}$. We show how to shift this application of (EW) below other rule applications until the statement is satisfied for such application. Three cases can arise:

1. S is the active component in the premise of an application of a rule (r). The conclusion of (r) is simply obtained by applying (EW) to G possibly multiple times.

2. S is a context component in the premise of an application of a one-premise rule (r). The (EW) is simply shifted below (r).

3. S occurs actively inside the queues of (EW) above all the premises of an application of a rule (r). We remove all the applications of (EW) with active component S in the queues and apply (r) with one context component less, followed by (EW).

The termination of the procedure follows from the fact that $\mathcal{D}$ is finite and that 1–3 always reduce the number of rules different from (EW) occurring below the (EW) applications. □

3.3 Applications of the embeddings

We just provided constructive transformations from hypersequent derivations to 2-system derivations and back. These transformations show that the two seemingly different proof frameworks have the same expressive power, but the embeddings are not only interesting for their conceptual outcomes, they also enable us to prove further technical results both concerning 2-systems and hypersequents.

3.3.1 Applications for 2-systems

The benefits of the embeddings with respect to 2-systems include: (i) new cut-free 2-systems, (ii) analyticity proofs, and (iii) locality of derivations using the hypersequent notation.

The points (i) and (ii) rely on the method in [CGT08] to transform propositional Hilbert axioms in the language of full Lambek calculus into suitable hypersequent rules, see Section 2.3.

Ad (i): the method in [Neg16] rewrites generalized geometric formulae in the class GA_1, see Section 2.1, into analytic 2-systems. As observed in [Neg16], formulae in GA_1 need not contain quantifier alternations; indeed there are purely propositional axioms that are in GA_1 but not in GA_0. Notice that the propositional axioms in GA_1 are strictly contained in the class $\mathcal{P}_3$ of [CGT08]. For the strictness of the inclusion, consider the excluded middle law $\mathrm{EM} = \neg A \vee \neg\neg A$. If we define, as usual, $\neg A$ as $A \to \bot$, this axiom belongs to $\mathcal{P}_3$ but not to GA_1. While the method in [Neg16] does not apply to $\neg A \vee \neg\neg A$, we can define a 2-system by translating the hypersequent rule corresponding to the axiom (below left) into the equivalent 2-system (below right):

$$\frac{G \mid \Sigma, \Sigma' \Rightarrow}{G \mid \Sigma \Rightarrow \mid \Sigma' \Rightarrow} \qquad \dfrac{\dfrac{}{\Sigma \Rightarrow} \quad \dfrac{\Sigma, \Sigma' \Rightarrow}{\Sigma' \Rightarrow}}{\dfrac{\Gamma \Rightarrow \Pi \qquad \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi}}$$

Ad (*ii*): The analiticity proof in [Neg16] relies on the fact that the obtained 2-systems manipulate atomic formulae only; this is the case for labelled 2-systems capturing frame conditions, but it does not hold anymore when translating propositional axiom schemata. In spite of this, for any 2-system obtained applying the method in [Neg16] to a GA_1 propositional Hilbert axiom, we obtain an analytic 2-system if we translate it into a hypersequent rule, apply the *completion* procedure of [CGT08], and translate the result back into a 2-system.

Example 3.3.1. We show the transformation of a 2-system into an analytic 2-system. Consider the excluded middle law $\mathrm{EM} = A \vee \neg A \in \mathrm{GA}_1$. The method in [Neg16] transforms it into the 2-system below on the left, which is translated into the hypersequent rule below on the right by the procedure in Section 3.1:

$$\dfrac{\dfrac{A, \Gamma_1 \Rightarrow \Delta_1}{\Gamma_1 \Rightarrow \Delta_1}\vdots \quad \dfrac{\bot, \Gamma_2 \Rightarrow \Delta_2}{A, \Gamma_2 \Rightarrow \Delta_2}\vdots}{\dfrac{\Gamma \Rightarrow \Delta \qquad \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}} \qquad \dfrac{G \mid A, \Gamma_1 \Rightarrow \Delta_1 \quad G \mid \bot, \Gamma_2 \Rightarrow \Delta_2}{G \mid \Gamma_1 \Rightarrow \Delta_1 \mid A, \Gamma_2 \Rightarrow \Delta_2}$$

Using the results in [CGT08] we can *complete* the latter rule and obtain the analytic hypersequent rule below on the left, whose translation leads to the 2-system below on the right:

$$\dfrac{G \mid \Sigma, \Gamma_1 \Rightarrow \Pi_1}{G \mid \Gamma_1 \Rightarrow \Pi_1 \mid \Sigma, \Gamma_2 \Rightarrow \Pi_2} \qquad \dfrac{\dfrac{\Sigma, \Gamma_1 \Rightarrow \Pi_1}{\Gamma_1 \Rightarrow \Pi_1}\vdots \quad \dfrac{}{\Sigma, \Gamma_2 \Rightarrow \Pi_2}\vdots}{\dfrac{\Gamma \Rightarrow \Pi \qquad \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi}}$$

The analiticity of $\mathrm{L}\mathfrak{I}$ extended with the obtained system of rules follows from Theorem 3.2.5.

3.3.2 Applications for hypersequent calculi

We show below how to use the embeddings to reformulate hypersequent calculi as natural deduction systems. This is a gain in terms of structural simplicity, since the basic objects of natural deduction derivations are formulae, as opposed to hypersequents.

Such reformulation will be exploited in Chapters 4, 5, and 6. as a basis for defining type systems for concurrent λ-calculi and show the connection, suggested in [Avr91], between intermediate logics captured by cut-free hypersequent systems and parallel λ-calculi.

Our reformulation of hypersequent calculi as natural deduction systems is modular and simply obtained by adding to Gentzen's $\mathrm{N}\mathfrak{I}$ *higher-level rules* simulating hypersequent rules acting on several components. The transformation from hypersequent derivations into 2-systems enables this reformulation since 2-systems derivations are close to natural deduction derivations.

To present the transformation in a simple way, henceforth we consider hypersequent rules of the following form:

$$\frac{M_1 \qquad \dots \qquad M_k}{G \mid \Sigma^1_1, \dots, \Sigma^1_{n_1}, \Gamma_1 \Rightarrow \Pi_1 \mid \dots \mid \Sigma^k_1, \dots, \Sigma^k_{n_k}, \Gamma_k \Rightarrow \Pi_k} \ (Hr)$$

where G and all variables for multisets of formulae are schematic, for any $1 \leq i \leq k$, M_i is a (possibly empty) set of hypersequents of the form $G \mid \Delta^i_j, \Gamma_i \Rightarrow \Pi_i$, for some j, with $\Delta^i_j = \Sigma^p_q$ for some $1 \leq p \leq k$ and $1 \leq q \leq n_p$, and with Γ_i and Π_i non-empty.

These rules are obtained by applying the algorithm in [CGT08] to $\mathcal{P}_3$ formulae (cf. the grammar in Section 3.3.1) of the following form[2]:

$$((A^1_1 \wedge \dots \wedge A^1_{n_1}) \to (B^1_1 \vee \dots \vee B^1_{m_1})) \vee \dots \vee ((A^k_1 \wedge \dots \wedge A^k_{n_k}) \to (B^k_1 \vee \dots \vee B^k_{m_k}))$$

where A^i_j and B^i_j are schematic variables and $(B^i_1 \vee \dots \vee B^i_{m_i})$ is $\bot$ if $m_i = 0$. Henceforth we will refer to this formula as the axiom *associated* to the rule (Hr). As shown in [CGT08], H$\mathfrak{I}$ extended with (Hr) is *equivalent* to H$\mathfrak{I}$ extended with its associated axiom – that is, the relative derivability relations coincide.

Example 3.3.2. Examples of $\mathcal{P}_3$ formulae are the linearity axiom $\mathrm{Lin} = (A \to B) \vee (B \to A)$, see Example 2.2.1, the law of excluded middle EM, and the axioms $(Bck) = A_0 \vee (A_0 \to A_1) \vee \dots \vee (A_0 \wedge \dots \wedge A_{k-1} \to A_k)$ that, for $k \geq 1$, axiomatize intermediate logics characterised by relational models with k worlds. Also the formulae in [LE82b] for implicational logics and the disjunctive tautologies in [DK00] are of this form.

Since we want to translate hypersequent calculi into natural deduction calculi and we can use the embedding in Section 3.1 to transform a hypersequent rule into a 2-system, the only missing link is a translation from 2-systems to natural deduction rules. Nonetheless, in order to directly translate systems of rules into natural deduction rules with a similar structure, we need to generalize our notion of natural deduction calculus. In particular, we need a natural deduction rule to be able to discharge not only formulae, but also other rule applications. In our calculus we will have the usual rules, that can be unconditionally applied, but we will also have conditional rules, that can only be applied if some other rule discharges their applications later on. Using conditional rules and rule discharge, we will be able to simulate the non-local conditions of 2-systems. Natural deduction rules that can discharge other rule applications have been studied by Schroeder-Heister under the name of higher-level rules [SH14].

Before considering the general problem of translating 2-systems into natural deduction rules, we present an example.

Example 3.3.3. The hypersequent rule for the linearity axiom $\mathrm{Lin} = (B \to A) \vee (A \to B)$ below on the left is translated into the 2-system at the center. This 2-system is translated

[2] In the general case, $\mathcal{P}_3$ formulae correspond to hypersequent rules similar to Hr in shape but with more than one Δ^i_j in each premise.

into the natural deduction rule below on the right:

$$\frac{G \mid A, \Gamma_1 \Rightarrow \Pi_1 \qquad G \mid B, \Gamma_2 \Rightarrow \Pi_2}{G \mid B, \Gamma_1 \Rightarrow \Pi_1 \mid A, \Gamma_2 \Rightarrow \Pi_2} \qquad \frac{\dfrac{A, \Gamma_1 \Rightarrow \Pi_1}{B, \Gamma_1 \Rightarrow \Pi_1} \quad \dfrac{B, \Gamma_2 \Rightarrow \Pi_2}{A, \Gamma_2 \Rightarrow \Pi_2}}{\Gamma \Rightarrow \Pi \qquad\qquad \Gamma \Rightarrow \Pi} \qquad \frac{\dfrac{A}{B} \quad \dfrac{B}{A}}{C \qquad\quad C}$$

Using this rule, Lin can be derived as follows

$$\frac{\dfrac{\dfrac{\dfrac{[A]^1}{B}\,{}^{*}}{A \to B}\,{}^{1}}{(A \to B) \vee (B \to A)} \quad \dfrac{\dfrac{\dfrac{[B]^2}{A}\,{}^{*}}{B \to A}\,{}^{2}}{(A \to B) \vee (B \to A)}}{(A \to B) \vee (B \to A)}\,{}^{*}$$

where we signal rule application discharge by $*$.

The addition to N$\mathfrak{I}$ of the resulting natural deduction rule yields the calculus N$\mathfrak{G}$ for Gödel–Dummett logic discussed in Chapter 4.

In general, a hypersequent rule of the form (*Hr*) above is transformed by the embedding in Section 3.1 into the following 2-system:

$$\frac{\dfrac{M_1}{\Sigma_1^1, \ldots, \Sigma_{n_1}^1, \Gamma_1 \Rightarrow \Pi_1}\ Tr_1 \qquad\qquad \dfrac{M_k}{\Sigma_1^k, \ldots, \Sigma_{n_k}^k, \Gamma_k \Rightarrow \Pi_k}\ Tr_k}{\Gamma \Rightarrow \Pi \qquad\quad \ldots \qquad\quad \Gamma \Rightarrow \Pi}$$

which is translated into a natural deduction rule *Nr* of the following form:

$$\frac{\dfrac{A_1^1 \quad \ldots \quad A_{n_1}^1 \quad \overset{[B_1^1]}{\vdots}\, A_1 \quad \ldots \quad \overset{[B_{m_1}^1]}{\vdots}\, A_1}{A_1} \qquad\qquad \dfrac{A_1^k \quad \ldots \quad A_{n_k}^k \quad \overset{[B_1^k]}{\vdots}\, A_k \quad \ldots \quad \overset{[B_{m_k}^k]}{\vdots}\, A_k}{A_k}}{\overset{\vdots}{A} \qquad\quad \ldots \qquad\quad \overset{\vdots}{A}} \tag{3.1}$$

where A_j^i corresponds to Σ_j^i and B_j^i corresponds to $\bot$ if $M_i = \emptyset$ and to Δ_j^i if M_i is a non-empty set of hypersequents of the form $G \mid \Delta_j^i, \Gamma_i \Rightarrow \Pi_i$.

As we have seen in Example 3.3.3 and as we will see in Example 3.3.4, it is possible to simplify the form of an inference discharged by a higher-level rule when there is only one schematic variable B_j^i in that inference.

The higher-level natural deduction calculi that we use in the present work can be formalized by generalizing the definition of natural deduction calculus in Section 2.4 as follows.

Definition 3.3.1. A higher-level natural deduction calculus is a pair $(\mathbb{R}_u, \mathbb{R}_c)$ where $\mathbb{R}_u$ is a set of unconditional inference rules and $\mathbb{R}_c$ is a set of conditional inference rules. Any inference rule in $\mathbb{R}_u \cup \mathbb{R}_c$ has the form

$$\dfrac{\begin{array}{ccc}[\mathcal{S}_1] & & [\mathcal{S}_n] \\ \vdots & & \vdots \\ A_1 & \dots & A_n\end{array}}{A}$$

where $\mathcal{S}_1, \dots, \mathcal{S}_n$ are possibly empty sets containing formulae and rule applications.

A natural deduction derivation of a formula A can then be inductively defined as follows:

- A is a derivation of A with open assumption A;
- if
$$\dfrac{\begin{array}{ccc}[\mathcal{S}_1] & & [\mathcal{S}_n] \\ \vdots & & \vdots \\ A_1 & \dots & A_n\end{array}}{A} \in \mathbb{R}_u$$
and $\mathcal{D}_1, \dots, \mathcal{D}_n$ are derivations of $A_1, \dots, A_n$ with open assumptions $\mathcal{T}_1, \dots, \mathcal{T}_n$, respectively, then
$$\dfrac{\mathcal{D}_1 \quad \dots \quad \mathcal{D}_n}{A}$$
is a derivation of A with open assumptions $\bigcup_{i=1}^{n}(\mathcal{T}_i \setminus \mathcal{S}_i)$.
- if
$$\dfrac{\begin{array}{ccc}[\mathcal{S}_1] & & [\mathcal{S}_n] \\ \vdots & & \vdots \\ A_1 & \dots & A_n\end{array}}{A} \in \mathbb{R}_c$$
and $\mathcal{D}_1, \dots, \mathcal{D}_n$ are derivations of $A_1, \dots, A_n$ with open assumptions $\mathcal{T}_1, \dots, \mathcal{T}_n$, respectively, then
$$\dfrac{\mathcal{D}_1 \quad \dots \quad \mathcal{D}_n}{A}$$
is a derivation of A with open assumptions $\left\{\dfrac{A_1 \quad \dots \quad A_n}{A}\right\} \cup (\bigcup_{i=1}^{n}(\mathcal{T}_i \setminus \mathcal{S}_i))$.

Intuitively, the open assumptions of a higher-level natural deduction derivation might contain rule applications. Moreover, we distinguish between unconditional rules $\mathbb{R}_u$ and conditional rules $\mathbb{R}_c$. An unconditional rule discharges the open assumptions $\mathcal{S}_1, \dots, \mathcal{S}_n$ and do not add any new open assumption to the derivation. The only difference between $\mathbb{R}_u$ rules and usual natural deduction rules is that among the open assumptions discharged by an $\mathbb{R}_u$ rule there could be some rule applications. When we apply an $\mathbb{R}_c$ conditional

rule, on the other hand, we also add the rule itself as a new open assumption. For instance, the conditional rule applications

$$\dfrac{A_1^j \quad \ldots \quad A_{n_j}^j \quad \begin{array}{c}[B_1^j]\\ \vdots\\ A_j\end{array} \quad \ldots \quad \begin{array}{c}[B_{m_j}^j]\\ \vdots\\ A_j\end{array}}{A_j}$$

with $1 \leq j \leq k$ in the rule schema (3.1) above are listed among the open assumptions until some higher-level rule discharges them.

Example 3.3.4. The hypersequent rule below left for the law of excluded middle $\mathrm{EM} = A \vee \neg A$ – see Example 3.3.1 for the corresponding 2-system – translates into the natural deduction rule below right:

$$\dfrac{G \mid \Sigma, \Gamma_1 \Rightarrow \Pi_1}{G \mid \Gamma_1 \Rightarrow \Pi_1 \mid \Sigma, \Gamma_2 \Rightarrow \Pi_2} \qquad \dfrac{\begin{array}{c}A\\ \vdots\\ \bot\\ \vdots\\ B\end{array} \quad \begin{array}{c}[A]\\ \vdots\\ B\end{array}}{B}$$

We can derive EM using this rule as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{[A]^1}{\bot}\,{\scriptstyle *}}{\neg A}\,{\scriptstyle 1}}{A \vee \neg A} \quad \dfrac{[A]^*}{A \vee \neg A}}{A \vee \neg A}\,{\scriptstyle *}$$

We show now that a hypersequent rule (*Hr*) and the corresponding natural deduction rule *Nr* are equivalent, i.e. that $\vdash_{\mathrm{H}\mathfrak{J}+Hr} A$ if and only if $\vdash_{\mathrm{N}\mathfrak{J}+Nr} A$.

Theorem 3.3.1. H$\mathfrak{J}$ *extended with any hypersequent rule (Hr) is equivalent to* N$\mathfrak{J}$ *extended with its translated rule Nr.*

Proof. We show that if $\vdash_{\mathrm{H}\mathfrak{J}+Hr} A$ then $\vdash_{\mathrm{N}\mathfrak{J}+Nr} A$. Indeed a derivation of the axiom r_α associated to (*Hr*) is as follows:

$$\dfrac{\dfrac{\dfrac{[A_1^1 \wedge \ldots \wedge A_{n_1}^1]^1}{A_1^1} \quad \ldots \quad \dfrac{[A_1^1 \wedge \ldots \wedge A_{n_1}^1]^1}{A_{n_1}^1} \quad \dfrac{[B_1^1]^2}{B_1^1 \vee \ldots \vee B_{m_1}^1} \quad \ldots \quad \dfrac{[B_{m_1}^1]^2}{B_1^1 \vee \ldots \vee B_{m_1}^1}\,{\scriptstyle 2\,*}}{B_1^1 \vee \ldots \vee B_{m_1}^1}}{(A_1^1 \wedge \ldots \wedge A_{n_1}^1) \to (B_1^1 \vee \ldots \vee B_{m_1}^1)}\,{\scriptstyle 1}}{r_\alpha} \qquad \begin{array}{c}\\ \\ \vdots\\ r_\alpha\end{array}}{r_\alpha} \quad \ldots \quad r_\alpha\,{\scriptstyle *}$$

All hypotheses are derived as shown for the leftmost. The rest of the premises of the bottom-most inference are derived similarly using the implications

$$(A_1^2 \wedge \ldots \wedge A_{n_2}^2) \to (B_1^2 \vee \ldots \vee B_{m_2}^2)\ ,\ \ldots\ ,\ (A_1^k \wedge \ldots \wedge A_{n_k}^k) \to (B_1^k \vee \ldots \vee B_{m_k}^k)$$

The claim follows by the equivalence between r_α and (Hr) shown in [CGT08].

To show that if $\vdash_{\mathrm{N}\mathfrak{J}+Nr} A$ then $\vdash_{\mathrm{H}\mathfrak{J}+Hr} A$, we derive the rule Nr using the rules of $\mathrm{N}\mathfrak{J}$ and r_α. We can then easily exploit the equivalence between $\mathrm{H}\mathfrak{J}$ and $\mathrm{N}\mathfrak{J}$. Intuitively, we use conjunction and implication elimination to simulate the inferences discharged by Nr – top left part of the following derivation. Then we nest one disjunction elimination $\vee e$ for each disjunctive subformula of the axiom in order to discharge the implications used above, discharge the formulae B^i_j, and derive $A, A_1, \ldots, A_k$:

$$\dfrac{\alpha \qquad \dfrac{\dfrac{[(A^1_1 \wedge \ldots \wedge A^1_{n_1}) \to (B^1_1 \vee \ldots \vee B^1_{m_1})]^1 \qquad \dfrac{A^1_1 \quad \dfrac{A^1_2 \quad A^1_3 \wedge \ldots \wedge A^1_{n_1}}{A^1_2 \wedge \ldots \wedge A^1_{n_1}}}{A^1_1 \wedge \ldots \wedge A^1_{n_1}}}{B^1_1 \vee \ldots \vee B^1_{m_1}} \qquad \begin{matrix} [B^1_1]^2 \\ \vdots \\ A_1 \end{matrix} \qquad \begin{matrix} \vdots \\ A_1 \end{matrix}}{A_1} \vee e^2 \qquad \begin{matrix} \vdots \\ A \end{matrix}}{A} \vee e^1$$

The open hypotheses here are the formulae $A^1_1, \ldots, A^1_{n_1}, \ldots, A^k_1, \ldots, A^k_{n_k}$, which are exactly the hypotheses of Nr. The claim follows by the equivalence between r_α and (Hr) shown in [CGT08]. □

Table 3.1 displays the translation into natural deduction rules of the hypersequent rules that are shown in Table 2.3.

A natural deduction calculus for classical logic based on the first rule presented here is introduced in Section 4.1, where we also provide a computational interpretation and a normalization proof for such calculus. In Section 4.2 we introduce a calculus for Gödel–Dummett logic based on the second rule of Table 3.1. Also for this calculus we provide a computational interpretation and a normalization proof. The rules for $\mathbf{G}_k$ and $\mathbf{C}_k$, on the other hand, can be captured by the framework introduced in Chapter 6, which provide computational interpretations and normalization results for all the natural deduction calculi falling in their scope. Finally, a computational interpretation and normalization result for $\mathbf{BC}_k$ can be obtained as a corollary from a very easy generalization of the framework in Chapter 6, see Remark 6.4.

hypersequent rule	*natural deduction rule*	*logic*
$\dfrac{G \mid \Gamma, \Sigma \Rightarrow \Delta}{G \mid \Gamma \Rightarrow \mid \Sigma \Rightarrow \Delta}$	$\dfrac{\dfrac{A}{\bot}\ \ [A]}{B}$	**CL**
$\dfrac{G \mid \Gamma, B \Rightarrow \Delta \quad G \mid \Sigma, A \Rightarrow \Theta}{G \mid \Gamma, A \Rightarrow \Delta \mid \Sigma, B \Rightarrow \Theta}$	$\dfrac{\dfrac{A}{B}\quad\dfrac{B}{A}}{C}$	**GL**
$\dfrac{\{G \mid \Gamma_{j+1}, \Delta_j \Rightarrow \Pi_j\}_{j\in\{0,\ldots,k-1\}}}{G \mid \Gamma_0, \Delta_0 \Rightarrow \Pi_0 \mid \ldots \mid \Gamma_{k-1}, \Delta_{k-1} \Rightarrow \Pi_{k-1} \mid \Gamma_k \Rightarrow}$	$\dfrac{\dfrac{A_0}{A_1}\quad\dfrac{A_{k-1}}{A_k}\quad\dfrac{A_k}{\bot}}{B}$	$\mathbf{G}_k$
$\dfrac{\{G \mid \Gamma_{j+1}, \Delta_j \Rightarrow \Pi_j\}_{j\in\{1,\ldots,k-1\}} \quad G \mid \Gamma_1, \Delta_k \Rightarrow \Pi_k}{G \mid \Gamma_1, \Delta_1 \Rightarrow \Pi_1 \mid \ldots \mid \Gamma_k, \Delta_k \Rightarrow \Pi_k}$	$\dfrac{\dfrac{A_1}{A_2}\quad\dfrac{A_{k-1}}{A_k}\quad\dfrac{A_k}{A_1}}{B}$	$\mathbf{C}_k$
$\dfrac{\{G \mid \Gamma_i, \Gamma_j \Rightarrow \Delta_i\}_{i,j\in\{0,\ldots,k\},\, i\neq j}}{G \mid \Gamma_0 \Rightarrow \Delta_0 \mid \ldots \mid \Gamma_k \Rightarrow \Delta_k}$	see Table 3.2	$\mathbf{BW}_k$
$\dfrac{\{G \mid \Gamma_i, \Gamma_j \Rightarrow \Delta_i\}_{i\in\{0,\ldots,k-1\},\, i<j\leq k}}{G \mid \Gamma_0 \Rightarrow \Delta_0 \mid \ldots \mid \Gamma_{k-1} \Rightarrow \Delta_{k-1} \mid \Gamma_k \Rightarrow}$	see Table 3.2	$\mathbf{BC}_k$

Table 3.1: Hypersequent and corresponding natural deduction rules.

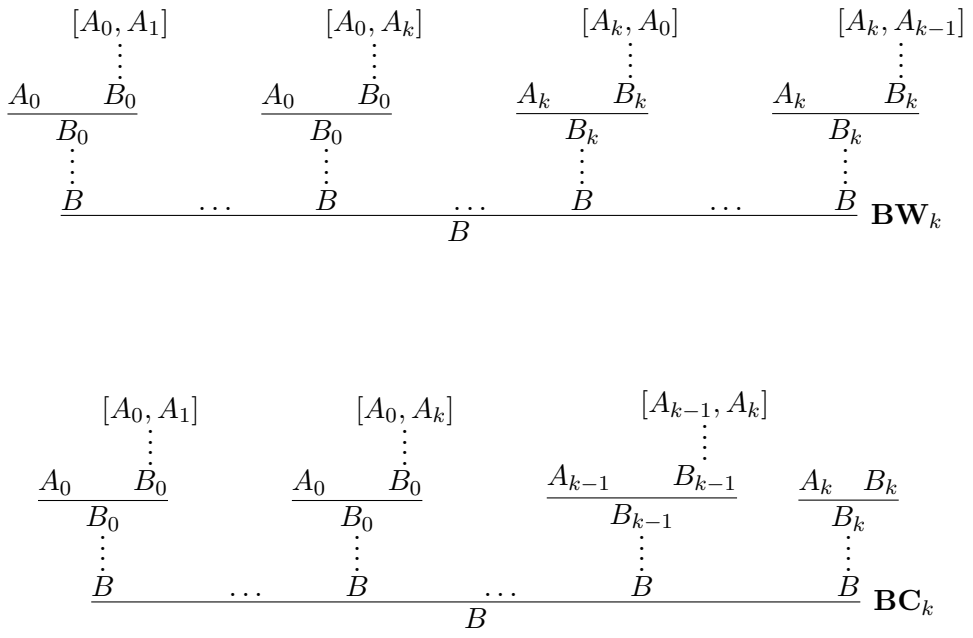


Table 3.2: Natural deduction rules for $\mathbf{BW}_k$ and $\mathbf{BC}_k$.

CHAPTER 4

Classical logic and Gödel–Dummett logic

In Chapter 3 we have seen how we can strip the hypersequent formalism of all its proof-theoretical structure in favor of non-local conditions granting exactly the same expressive power. The translation from hypersequents to 2-systems enables us to dispense with the hypersequent symbol |. The translation from 2-systems to natural deduction frees us also from the remaining sequent structure. It leaves us with intuitionistic proofs constructed using only simple inferences and related by non-local conditions, as discussed in Section 2.4.3 and shown at page 28. Intuitionistic natural deduction proofs are not only simple from a structural perspective, they can also be very naturally interpreted as computer programs through the Curry–Howard correspondence [How80]. Thus, the translations presented in the previous chapter gives us the right framework to try to confirm Avron's thesis on the concurrent nature of hypersequent proofs [Avr91] by providing computational interpretations for the intermediate logics that can be captured by hypersequent calculi.

If we apply the translations presented in Chapter 3 to suitable hypersequent calculi we obtain natural deduction calculi which are defined extending the intuitionistic natural deduction $\mathrm{N}\mathfrak{I}$ by non-local conditions acting across different subderivations, as detailed in Section 3.3.2. Since there is a very robust computational interpretation of $\mathrm{N}\mathfrak{I}$, which forms the purely intuitionistic backbone of our calculi, we only need an interpretation of the non-local conditions in terms of information exchange and we have all we need to define concurrent λ-calculi. This is what we will do in the present chapter, starting from a calculus for classical logic featuring a very simple communication mechanism and then presenting the calculus for Gödel–Dummett logic, which from a computational point of view can be seen as a generalization of the first one.

From a technical point of view, in the present chapter we will introduce concurrent

λ-calculi based on the proof systems for propositional classical logic (λ_{Cl}) and Gödel–Dummett logic (λ_{G}) obtained by the translation in Section 3.3.2, see Table 3.1. The channels of both calculi connect two processes, but while in λ_{Cl} the direction of the channel is fixed, in λ_{G} the channels can be used to transmit messages in both directions. We show, moreover, normalization results for both calculi and prove that the normal form proof terms enjoy the subformula property. Thus we provide the first computational interpretations of classical and Gödel–Dummett logic in terms of concurrent computation.

The present chapter is based on results and ideas published in [ACG17] and [ACG18]. These results are refined and extended here. We present λ_{Cl} in Section 4.1. Its type system and reduction rules are described in Section 4.1.1, we show that each normal λ_{Cl}-term enjoys the subformula property in Section 4.1.2, in Section 4.1.3 we prove a normalization result for λ_{Cl}, and in Section 4.1.4 we present results and examples concerning the expressiveness of λ_{Cl}. Section 4.2 is about λ_{G}. We present the natural deduction system in Section 4.2.1 and discuss its reductions in Section 4.2.2. In Section 4.2.3 we introduce the calculus, in Section 4.2.4 we show that normal λ_{G}-terms enjoy the subformula property, and in Section 4.2.5 we prove a normalization result for λ_{G}. We study the expressive power of λ_{G} in Section 4.2.6. In Section 4.3 we compare λ_{Cl} and λ_{G} with each other and with related calculi.

4.1 One-way communication: λ_{Cl}

We present the simplest of the concurrent calculi studied here: λ_{Cl}. The calculus λ_{Cl} is an extension of λ-calculus by the $\|_a$ operator, a parallelism operator and binder for communication variables a. These communication variables behave as higher-order communication channels and enable parallel processes to transmit sub-processes. The elementary communication structure of such calculus is the simplest kind of communication network, namely a unidirectional link between two processes:

Figure 4.1: Representation of a λ_{Cl} channel.

This means that if a processes u contains the negative channel variable $\overline{a} : \neg A$ and a process v contains its dual $a : A$, and they occur in a term $u \|_a v$, then u can transmit to v the argument of the variable $\overline{a} : \neg A$. Since an argument of $\overline{a} : \neg A$ has type A and the reception point inside the term v is $a : A$, the transmission does not violate the type assignment.

The type assignment rule for the $\|_a$ operator is

$$\dfrac{\begin{array}{cc}[a^{\neg A} : \neg A] & [a^{A} : A] \\ \vdots & \vdots \\ u : B & v : B\end{array}}{u \parallel_a v : B}\ (\mathrm{EM})$$

and is based on the excluded middle law $\mathrm{EM} = \neg A \vee A$. If the channel variables $\overline{a}$ and a have dual types $\neg A$ and A respectively, when we introduce the operator $\|_a$, we can bind all their occurrences. This corresponds to discharging the assumptions $\neg A$ and A of the logical derivation. From a proof theoretical point of view, the rule corresponds to a proof by case distinction on the truth of A: if we can prove B both assuming A and assuming $\neg A$, then we can infer B discharging the assumptions A and $\neg A$.

The calculus λ_{Cl} is a concurrent computational interpretation of classical logic. As mentioned in Section 2.4.2, we provide here a different computational interpretation of the logical system adopted for λ_{exn}. A major proof-theoretical difference between λ_{exn} and λ_{Cl} lies in the fact that λ_{exn} normal proof terms do not necessarily correspond to analytic proofs, while λ_{Cl} normal proof terms do.

4.1.1 The type system of λ_{Cl} and its reduction rules

Considering first the natural deduction from a purely proof theoretical perspective, we show that $\mathrm{N}\mathfrak{I} + (\mathrm{EM})$ for

$$(\mathrm{EM}) \quad = \quad \dfrac{\begin{array}{cc}[\neg A] & [A] \\ \vdots & \vdots \\ B & B\end{array}}{B}$$

is sound and complete with respect to classical logic.

Notice that this is equivalent to the higher-level rule

$$\dfrac{\begin{array}{cc}\dfrac{A}{\bot} & [A] \\ \vdots & \vdots \\ B & B\end{array}}{B}$$

defined in Section 3.3.2. Since we are using this rule here to type terms of the form $u \parallel_a v$, (EM) has the benefit of displaying a more direct correspondence with the types of $a : \neg A$ in u and $a : A$ in v, see for example Table 4.1. The proof-theoretical gain of the higher-level version, on the other hand, is that it is a purely structural rule – namely, a rule that makes no reference to any logical connective – and hence admits a simpler formulation of the subformula property, see *communication kind* in Definition 4.1.9. Even though the soundness and completeness of $\mathrm{L}\mathfrak{I}$ extended by the higher-level rule above with respect to **CL** are guaranteed by Theorem 3.3.1, we prove these results here directly for $\mathrm{L}\mathfrak{I} + (\mathrm{EM})$.

Theorem 4.1.1 (Soundness and completeness)**.** *For any set* Γ *of formulae and formula* A*,* $\Gamma \vdash_{\mathrm{N}\mathfrak{J}+(\mathrm{EM})} A$ *if and only if* $\Gamma \vdash_{\mathbf{CL}} A$*.*

Proof. The calculus $\mathrm{N}\mathfrak{J}$ is strongly equivalent to any Hilbert calculus for intuitionistic logic – see for example Chapter 2 of [TS96] – and we can define classical logic as the extension of intuitionistic logic by the axiom schema EM. Hence for the left to right direction, we show that we can simulate any instance of the rule (EM) using instances of the EM axiom in $\mathrm{N}\mathfrak{J}$ as follows:

$$\dfrac{\neg A \vee A \qquad \begin{array}{c}[\neg A]^1 \\ \vdots \\ C\end{array} \qquad \begin{array}{c}[A]^1 \\ \vdots \\ C\end{array}}{C}\ {}_1$$

While for the right to left direction, we show that we can derive any instance of the EM axiom as follows:

$$\dfrac{\dfrac{[\neg A]^1}{\neg A \vee A} \qquad \dfrac{[A]^1}{\neg A \vee A}}{\neg A \vee A}$$

□

We exploit the definability of $A \vee B$ in classical logic as $(A \to \bot) \to (B \to \bot) \to \bot$ and treat $\vee$ as a defined connective. See Definition 4.1.16 and Proposition 4.1.12 in Section 4.1.5 for the details.

Definition 4.1.1 (Natural deduction calculus $\mathrm{N}\mathfrak{Cl}$)**.** The natural deduction calculus $\mathrm{N}\mathfrak{Cl}$ extends $\mathrm{N}\mathfrak{J}^{\to}_{\wedge\bot}$ by the (EM) rule.

We have now all the elements to introduce λ_{Cl}, a typed concurrent λ-calculus for Classical logic. The calculus λ_{Cl} extends simply typed λ-calculus [How80] by a parallel operator that provides a computational interpretation for the rule (EM).

Definition 4.1.2 (Terms of λ_{Cl})**.** The terms of λ_{Cl} are defined by the rules for simply typed λ-calculus in Table 2.5 and by the rules in Table 4.1.

The parallelism operator supporting communication is introduced by the rule (EM) while the contraction rule (*contr*) is used to define parallel terms that do not communicate, see Definition 4.1.3. From a proof-theoretical perspective, (EM) combines the behavior of the hypersequent rule (EC) and of the hypersequent rule for $A \vee \neg A$ in Table 2.3. On the other hand, (*contr*) corresponds to an application of the rule (EC) alone.

We provide the essential notation, definitions and terminology. Proof terms may contain variables $x_0^A, x_1^A, x_2^A, \ldots$ of type A for every formula A. For clarity, the variables introduced by the (EM) rule will be often denoted with letters $a, b, c, \ldots$ but they are

$$\dfrac{\begin{array}{cc}[a^{\neg A}:\neg A] & [a^{A}:A]\\ \vdots & \vdots\\ u:B & v:B\end{array}}{u \parallel_a v : B}\ (\mathrm{EM}) \qquad \dfrac{u:A \quad v:A}{u \parallel v : A}\ (\mathit{contr})$$

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

Table 4.1: Type assignments for λ_{Cl}.

not in a syntactic category apart. A variable x^A that occurs in a term of the form $\lambda x^A u$ is called λ-*variable* and a variable a that occurs in a term $u \parallel_a v$ is called *channel* or *communication variable* and represents a private communication channel between the parallel processes u and v. We adopt the convention that variables $a^{\neg A}$ and a^A will be respectively denoted as $\overline{a}$ and a, where unambiguous.

Free and bound variables of a term are defined as usual. For the new term $u \parallel_a v$, all free occurrences of a in u and v are bound in $u \parallel_a v$. We assume the standard renaming rules and α-equivalences that are used to avoid the capture of variables in the reductions.

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

The reduction rules of λ_{Cl} are presented in Table 4.2. They consist of the usual simply typed λ-calculus reductions, instances of $\vee$ permutations adapted to the $\parallel$ operator, and new communication reductions: *basic cross reductions*, *simplification reductions* and *cross reductions*.

Since we are dealing with a Curry–Howard correspondence, every reduction rule of λ_{Cl} corresponds to a reduction for the natural deduction calculus $\mathrm{N}\mathfrak{I}^{\rightarrow}_{\wedge\perp} + (\mathrm{EM})$.

The main reductions for the (EM) rule are basic cross reductions and cross reductions, and are formally introduced in Table 4.2. Before discussing their computational aspects, we explain them from a proof-theoretical point of view. Basic cross reductions correspond to the following transformation of natural deduction derivations

$$\dfrac{\dfrac{[\neg A]\quad \overset{\delta}{A}}{\begin{array}{c}\perp\\ \vdots\\ C\end{array}} \qquad \begin{array}{c}[A]\\ \vdots\\ C\end{array}}{C} \quad \mapsto_{\mathsf{c}} \quad \begin{array}{c}\overset{\delta}{A}\\ \vdots\\ C\end{array}$$

where no assumption of δ is discharged below $\perp$ but above C – we address the general case below by using cross reductions. Intuitively, the displayed instance of (EM) might be hiding some redex that should be reduced. The reduction precisely exposes this

intuitionistic potential redex and we are thus able to reduce it. More instances of $\neg A$ and A might occur in the respective branches. This, in combination with the contraction rule (see Table 4.1), gives rise to races and broadcasting, as explained in the computational interpretation below. Cross reductions correspond to

$$\dfrac{\dfrac{[\neg A]^1 \quad \overset{[\Gamma]}{\overset{\delta}{A}}}{\overset{\vdots}{\underset{C}{\bot}}} \qquad \overset{[A]^1}{\underset{C}{\vdots}}}{C}\,{}_1 \quad \mapsto_{\mathsf{c}} \quad \dfrac{\dfrac{[\neg \bigwedge \Gamma]^2 \quad \overset{[\Gamma]}{\overline{\bigwedge \Gamma}}}{\overset{\vdots}{\underset{C}{\bot}}} \qquad \overset{[A]^1}{\underset{C}{\vdots}}}{C}\,{}_1 \qquad \dfrac{\overset{[\bigwedge \Gamma]^2}{\overline{\Gamma}} \quad \overset{\delta}{A}}{\underset{C}{\vdots}}\,{}_2}{C}$$

As in the previous case, in the derivation on the right we use δ to prove all occurrences of the assumptions A, but now we also need to discharge the assumptions Γ open in δ in the rightmost branch but discharged in the derivation on the left below $\bot$ and above C. This is achieved by a new (EM) rule application 2 to the conjunction $\bigwedge \Gamma$ of such assumptions. Accordingly, we use the inference $\dfrac{\neg \bigwedge \Gamma \quad \bigwedge \Gamma}{\bot}$ in the leftmost branch. We keep the original instance 1 of (EM) in order to discharge remaining occurrences of $\neg A$ on which the derivation of C from $\bot$ might still depend. Thus, the central branch of the resulting proof is just a duplicate. This reduction eliminates one occurrence of $\neg A$ and can be reiterated to remove all occurrences of $\neg A$ discharged by rules like 1 applied in the derivation. The cost paid to obtain this is the multiplication of some branches of the derivation, but simplification reductions of the form

$$\dfrac{\overset{\delta}{C} \qquad \overset{[A]^1}{\underset{C}{\vdots}}}{C}\,{}_1 \quad \mapsto_{\mathsf{c}} \quad \overset{\delta}{C}$$

where no occurrence of $\neg A$ is discharged by 1 in δ, can be later used to eliminate the unnecessary duplicates.

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

Definition 4.1.3 (Simple parallel term)**.** A *simple parallel term* is a λ_{Cl}-term $t_1 \parallel \ldots \parallel t_n$, where each t_i, for $1 \leq i \leq n$, is a simply typed λ-term.

We define as usual the notion of context $\mathcal{C}[\,]$ as the part of a proof term that surrounds a hole, represented by some fixed variable. In the expression $\mathcal{C}[u]$ we denote a particular occurrence of a subterm u in the whole term $\mathcal{C}[u]$. Contexts will be used to define communication reduction rules. A context will represent a process and the context hole will contain the channel application that is going to be used to communicate.

Definition 4.1.4 (Context)**.** A *context* $\mathcal{C}[\,]$ is a λ_{Cl}-term with some fixed variable $[\,]$ occurring exactly once.

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

Definition 4.1.5 (Simple and Simple Parallel Context)**.**

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

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

Definition 4.1.6 (Multiple substitution)**.** Let u be a proof term, $\boldsymbol{x} = x_1^{A_1}, \ldots, x_n^{A_n}$ a sequence of variables and $v : A_1 \wedge \ldots \wedge A_n$. The substitution $u^{v/\boldsymbol{x}} := u[v\,\pi_1/x_1^{A_1} \ldots\ v\,\pi_n/x_n^{A_n}]$ replaces each variable $x_i^{A_i}$ of any term u with the ith projection of v.

We now discuss the classes of computational reductions that will be formally introduced in Table 4.2.

Intuitionistic reductions These are the usual computational rules for the simply typed λ-calculus, see Section 2.4.1, representing the operations of applying a function and taking a component of a pair [GLT89]. From the logical point of view, they are the standard Prawitz reductions [Pra71] for $\mathrm{N}\mathfrak{I}^{\rightarrow}_{\wedge\perp}$.

Basic cross reductions We can trigger a basic cross reduction for a term $\mathcal{C}[\overline{a}\ u] \parallel_a \mathcal{D}$ only when the free variables of u are also free in $\mathcal{C}[\overline{a}\ u]$. The term u might represent executable code or data and directly replace all occurrences of the channel endpoint a in $\mathcal{D}$. We can only transmit a term u that does not contain a, because otherwise moving it to the right of the $\parallel_a$ operator would result in an unsound reduction. In case there is only one sender and one receiver, the reduction is

$$\mathcal{C}[\overline{a}\ u] \parallel_a \mathcal{D} \ \mapsto_{\mathtt{c}}\ \mathcal{D}[u/a]$$

In general, $\mathcal{C}[\overline{a}\ u]$ has the shape

$$\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[\overline{a}\ u] \parallel \ldots \parallel \mathcal{C}_n$$

where more than one process might have a message to send. In this case, the reduction requires a *race* among the processes $\mathcal{C}_j$ that contain some message $\overline{a}\, u_j$. These processes, indeed, compete for the possibility to transmit their message to $\mathcal{D}$. The sender $\mathcal{C}_i[\overline{a}\ u]$ is selected non-deterministically and communicates its message to $\mathcal{D}$:

$$(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[\overline{a}\ u] \parallel \ldots \parallel \mathcal{C}_n) \parallel_a \mathcal{D} \ \mapsto_{\mathtt{c}}\ \mathcal{D}[u/a]$$

Since the receiving term $\mathcal{D}$ exhausts all its channels $a : A$ to receive u, we remove all processes containing $\overline{a} : \neg A$ and obtain a term without instances of the channel a. We

also point out that $\mathcal{D}$ is an arbitrary term, so it may well be a sequence of parallel process $\mathcal{D}_1 \parallel \ldots \parallel \mathcal{D}_m$. In this case

$$\mathcal{D}[u/a] = \mathcal{D}_1[u/a] \parallel \ldots \parallel \mathcal{D}_m[u/a]$$

and $\mathcal{C}_i[\overline{a}\ u]$ *broadcasts* its message u to all the process $\mathcal{D}_1, \ldots, \mathcal{D}_m$.

Simplification reductions These reductions enable us to eliminate useless communication channels and to remove unnecessary duplicates generated by cross reductions and permutation reductions. Suppose, for example, that the process u in $u \parallel_a v$ does not contain occurrences of the channel a while the process v does. This intuitively means that v is waiting to communicate with u, but the communication is impossible. If this is the case, we simply apply the reduction $u \parallel_a v \mapsto_{\mathsf{c}} u$ in order to remove the useless channel a and keep only the process u which does not need it.

Cross reductions These reductions model λ_{Cl} communication mechanism between parallel processes in its full generality. In order to apply a cross reduction to a term

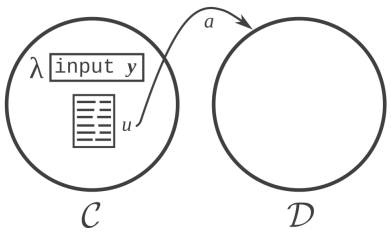
$$(\ldots \parallel \mathcal{C}[\overline{a}\ u] \parallel \ldots) \parallel_a \mathcal{D}$$

several conditions have to be met. These conditions are both natural and needed for the termination of computations. First, we require that the type of a violates the subformula property – which is formalized by the notion of communication complexity, see Definition 4.1.9 below. We use here the violation of the subformula property as a criterion for terminating the computation since, as shown by the reduction (4.1) at page 70, unrestricted cross reductions might result in reduction loops. We also require $(\ldots \parallel \mathcal{C}[\overline{a}\ u] \parallel \ldots)$ to be a normal simple parallel term. The normality of the term guarantees that the communication is unavoidable because the rest of the computation has finished. Finally, we require the variable a to be rightmost because, as for basic cross reductions, if the term u contained a, it could not be moved to the right of the $\parallel_a$ operator.

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

$$(\ldots \parallel \mathcal{C}[\overline{a}\ u] \parallel \ldots) \parallel_a \mathcal{D} \mapsto_{\mathsf{c}} (\mathcal{C}[\overline{b}\ \langle \boldsymbol{y} \rangle] \parallel_a \mathcal{D}) \parallel_b \mathcal{D}[u^{b/\boldsymbol{y}}/a]$$

Here, we have a term u to be transmitted to the right in order to replace all occurrences of a in $\mathcal{D}$:



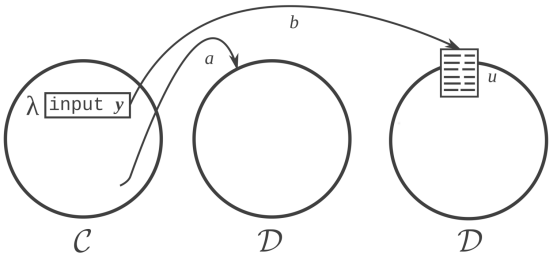
If u did not depend on the computational environment $\mathcal{C}[\ ]$ we could use a basic cross reduction and send u as it is. In general, though, this is not possible. The problem is that the free variables $\boldsymbol{y}$ of u which are bound in $\mathcal{C}[a\,u]$ by some λ cannot be permitted to become free; otherwise, the connection between the binders $\lambda\,\boldsymbol{y}$ and the occurrences of the variables $\boldsymbol{y}$ would be lost and they could be no more replaced by actual values when the inputs for the $\lambda\,\boldsymbol{y}$ are available. For example, we could have $u = v\,\boldsymbol{y}$ and

$$\mathcal{C}[a\,u] = w\,(\lambda\boldsymbol{y}\,a\,(v\,\boldsymbol{y}))$$

and the transformation

$$(\ldots \parallel w\,(\lambda\boldsymbol{y}\,a\,(v\,\boldsymbol{y}))\parallel \ldots)\parallel_a \mathcal{D} \;\mapsto_{\mathsf{c}}\; (\ldots \parallel w\,(\lambda\boldsymbol{y}\,a\,(v\,\boldsymbol{y}))\parallel \ldots)\parallel_a \mathcal{D}[v\,\boldsymbol{y}/a]$$

would be computationally wrong since the term $v\,\boldsymbol{y}$ will have no access to the actual values of the variables $\boldsymbol{y}$ when they will become available to $\lambda\boldsymbol{y}\,a\,(v\,\boldsymbol{y})$. Hence, we need to restore the access of u to its original computational environment and we do it by a new channel b:



The correct reduction for the previous example would then be

$$(\ldots \parallel w\,(\lambda\boldsymbol{y}\,a\,(v\,\boldsymbol{y}))\parallel \ldots)\parallel_a \mathcal{D} \;\mapsto_{\mathsf{c}}\; ((w\,(\lambda\boldsymbol{y}\,b\boldsymbol{y}))\parallel_a \mathcal{D})\;\parallel_b\; \mathcal{D}[v\,b/a]$$

where the channel b can be used to transmit the values of $\boldsymbol{y}$ from the process $w\,(\lambda\boldsymbol{y}\,b\boldsymbol{y})$ to the process $\mathcal{D}[v\,b/a]$ when they are available. In general, in the resulting term $(\mathcal{C}[\overline{b}\ \langle\boldsymbol{y}\rangle]\parallel_a \mathcal{D})\parallel_b \mathcal{D}[u^{b/\boldsymbol{y}}/a]$ the substitution $b/\boldsymbol{y}$ guarantees that when the values of $\boldsymbol{y}$ will be available in $\mathcal{C}[\overline{b}\ \langle\boldsymbol{y}\rangle]$, they will be sent to u through b.

In the result of the cross reduction the process $\mathcal{D}$ is *cloned* because $\mathcal{C}[\overline{a}\ u]$ might have other messages to transmit through a. Thus a behaves as a *replicated input*, as for example in [CP10a], replicated input is coded by the bang operator of linear logic:

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

These problems are typical of *process migration*, and can be solved by the concepts of *code mobility* [FPV98] and *closure* [Lan64]. Informally, code mobility is defined as the capability to dynamically change the bindings between code fragments and the locations where they are executed. Indeed, in order to be executed, a piece of code needs

a computational environment and its resources, like data, program counters or global variables. In our case the context $\mathcal{C}[\ ]$ is the computational environment or *closure* of the process u and the variables $\boldsymbol{y}$ are the resources it needs. Now, moving a process outside its environment always requires extreme care: the bindings between a process and the environment resources must be preserved. This is the task of the *migration mechanisms*, which allow a migrating process to resume correctly its execution in the new location. Our migration mechanism creates a new communication channel b between the programs that have been exchanged.

Example 4.1.1 (A concrete cross reduction)**.** If we add a type for natural numbers and a constant for addition in our calculus we can construct the concrete term $c\,(\lambda y\, a(\lambda x\,(y + (2 + x))))\ \|_a\ a\,3$ where a and c are channel variables. Suppose moreover that this term occurs in an environment $\mathcal{E}$. If we apply a cross reduction, we transmit the message $(\lambda x\,(y + (2 + x)))[b/y] = \lambda x\,(b + (2 + x))$ through the channel a

$$\mathcal{E}[c\,(\lambda y\, a(\lambda x\,(y+(2+x))))\ \|_a\ a\,3] \qquad \mapsto_{\mathsf{c}} \qquad \mathcal{E}[(c\,(\lambda y\, by)\ \|_a\ a\,3)\ \|_b\ (\lambda x\,(b+(2+x)))\,3]$$

another reduction then simplifies the resulting term into

$$\mapsto_{\mathsf{c}} \qquad \mathcal{E}[c\,(\lambda y\, by)\ \|_b\ (\lambda x\,(b + (2 + x)))\,3]$$

The new communication channel b has been introduced here because the message $\lambda x\,(y + (2 + x))$ still depends on the binder λy. Now we can normalize the process on the right without waiting for the value of y and we have

$$\mapsto_{\mathsf{c}} \qquad \mathcal{E}[c\,(\lambda y\, by)\ \|_b\ b + 5]$$

Suppose then that a communication from a process in $\mathcal{E}$ replaces c with the term $\lambda x\, x1$ and yields $\mathcal{E}'[(\lambda x\, x1)(\lambda y\, by)\ \|_b\ b+5]$. The following series of reductions is then triggered:

$$\mathcal{E}'[(\lambda x\, x1)(\lambda y\, by)\ \|_b\ b + 5] \quad \mapsto_{\mathsf{c}}^* \qquad \mathcal{E}'[b1\ \|_b\ b + 5]$$

Finally, we can transmit the value 1 of y through b and conclude the computation:

$$\mathcal{E}'[b1\ \|_b\ b + 5] \qquad \mapsto_{\mathsf{c}} \qquad \mathcal{E}'[1 + 5] \quad \mapsto_{\mathsf{c}}^* \qquad \mathcal{E}'[6]$$

Parallel operator permutations The only permutations for $\|$ that are not adaptations of the standard $\vee$-permutation are

$$(u\ \|_a\ v)\ \|_b\ w \mapsto_{\mathsf{c}} (u\ \|_b\ w)\ \|_a\ (v\ \|_b\ w) \quad \text{ and } \quad w\ \|_b\ (u\ \|_a\ v) \mapsto_{\mathsf{c}} (w\ \|_b\ u)\ \|_a\ (w\ \|_b\ v)$$

These address the *scope extrusion* issue of private channels. For instance, let us consider the term

$$(v\ \|_a\ \mathcal{C}[b\,a])\ \|_b\ w$$

Here the process $\mathcal{C}[b\,a]$ wishes to send the channel a to w along the channel b, but this is not possible since the channel a is private. This issue is solved in π-calculus using the

congruence $\nu a(P \mid Q) \mid R \equiv \nu a(P \mid Q \mid R)$, provided that a does not occur in R, condition that can always be satisfied by α-conversion. Our logical system requires a different solution, which is not just permuting w inward but also duplicating it:

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

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

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

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

Example 4.1.2 ($\parallel_a$ **in** λ_{Cl} **and** $\mid$ **in** π**-calculus**)**.** A private channel $u \parallel_a v$ is rendered in the π-calculus [Mil92, SW03] by the restriction operator ν, as $\nu a\,(u \mid v)$. Recall that π-calculus term $u \mid v$ represents two processes that run in parallel. The corresponding λ_{Cl}-term is $u \parallel v$ containing the parallelism operator $\parallel$ that does not support any communication channel.

We provide now the last definitions needed to formally define the reduction rules of λ_{Cl}. We start with the notion of *strong subformula*. This is key to define the communication complexity, Definition 4.1.9 below, which is in turn essential to define the conditions of the communication reductions of λ_{Cl}.

Definition 4.1.7 (Prime formulae and factors [Kri90])**.** A formula is said to be *prime* if it is not a conjunction. Every formula is a conjunction of prime formulae, called *prime factors*.

Definition 4.1.8 (Strong subformula)**.** B is said to be a *strong subformula* of a formula A, if B is a proper subformula of some prime proper subformula of A.

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

Proposition 4.1.2 (Characterization of strong subformulae)**.** *If B is a strong subformula of A:*

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

Proof.

- Suppose $A = A_1 \wedge \ldots \wedge A_n$, with $n > 0$ and $A_1, \ldots, A_n$ are prime. Any prime proper subformula of A is a subformula of one among $A_1, \ldots, A_n$, so B must be a proper subformula of one among $A_1, \ldots, A_n$.
- Suppose $A = C \to D$. Any prime proper subformula $\mathcal{X}$ of A is first of all a subformula of C or D. Assume now $C = C_1 \wedge \ldots \wedge C_n$ and $D = D_1 \wedge \ldots \wedge D_m$, with $C_1, \ldots, C_n, D_1, \ldots, D_m$ prime. Since $\mathcal{X}$ is prime, it must be a subformula of one among $C_1, \ldots, C_n, D_1, \ldots, D_m$ and since B is a proper subformula of $\mathcal{X}$, it must be a proper subformula of one among $C_1, \ldots, C_n, D_1, \ldots, D_m$.

□

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

$$\lambda y^B\, a^{\neg B}\, y \parallel_a x^{B \to \neg B}\, a^B \;\mapsto_{\mathsf{c}}\; (\lambda y\, b\, y \parallel_a x\, a) \parallel_b x\, b \;\mapsto_{\mathsf{c}}\; \lambda y\, b\, y \parallel_b x\, b \tag{4.1}$$

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

Definition 4.1.9 (Communication complexity)**.** Let $u \parallel_a v : A$ a proof term with free variables $x_1^{A_1}, \ldots, x_n^{A_n}$. Assume that $\overline{a} : \neg B$ occurs in u and $a : B$ occurs in v.

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

We recall the notion of stack, corresponding to Krivine stack [Kri09] and known as *continuation* because it embodies a series of tasks that wait to be carried out. A stack represents, from the logical perspective, a series of elimination rules; from the λ-calculus perspective, a series of either operations or arguments.

Definition 4.1.10 (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 $\sigma_i = t$, with t proof term or $\sigma_i = \pi_j$, with $j \in \{0, 1\}$. Now on we will denote the *empty sequence* with ϵ and

Intuitionistic reductions

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

Parallel Operator Permutations

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

$$(u \parallel_a v)\xi \mapsto_{\mathsf{c}} u\xi \parallel_a v\xi \text{ if } \xi \text{ is a one-element stack and } a \text{ does not occur free in } \xi$$

$$w(u \parallel v) \mapsto_{\mathsf{c}} wu \parallel wv \qquad (u \parallel v)\xi \mapsto_{\mathsf{c}} u\xi \parallel v\xi \qquad \lambda x^A\, (u \parallel_a v) \mapsto_{\mathsf{c}} \lambda x^A\, u \parallel_a \lambda x^A\, v$$

$$\langle u \parallel_a v,\, w\rangle \mapsto_{\mathsf{c}} \langle u, w\rangle \parallel_a \langle v, w\rangle \qquad \langle w,\, u \parallel_a v\rangle \mapsto_{\mathsf{c}} \langle w, u\rangle \parallel_a \langle w, v\rangle \quad \lambda x^A\, (u \parallel v) \mapsto_{\mathsf{c}} \lambda x^A\, u \parallel \lambda x^A\, v$$

$$\langle u \parallel v,\, w\rangle \mapsto_{\mathsf{c}} \langle u, w\rangle \parallel \langle v, w\rangle \qquad \langle w,\, u \parallel v\rangle \mapsto_{\mathsf{c}} \langle w, u\rangle \parallel \langle w, v\rangle$$

$$(u \parallel_a v) \parallel_b w \mapsto_{\mathsf{c}} (u \parallel_b w) \parallel_a (v \parallel_b w)$$

if the communication complexity of b is greater than 0

$$w \parallel_b (u \parallel_a v) \mapsto_{\mathsf{c}} (w \parallel_b u) \parallel_a (w \parallel_b v)$$

if the communication complexity of b is greater than 0

$$(u \parallel v) \parallel_b w \mapsto_{\mathsf{c}} (u \parallel_b w) \parallel (v \parallel_b w)$$

if the communication complexity of b is greater than 0

$$w \parallel_b (u \parallel v) \mapsto_{\mathsf{c}} (w \parallel_b u) \parallel (w \parallel_b v)$$

if the communication complexity of b is greater than 0

Communication Reductions

Basic cross reductions $\mathcal{C}[\overline{a}\ u] \parallel_a \mathcal{D} \mapsto_{\mathsf{c}} \mathcal{D}[u/a]$

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

Simplification reductions $u \parallel_a v \mapsto_{\mathsf{c}} u$ if a does not occur in u

$u \parallel_a v \mapsto_{\mathsf{c}} v$ if a does not occur in v

Cross reductions

$$(\ldots \parallel \mathcal{C}[\overline{a}\ u] \parallel \ldots) \parallel_a \mathcal{D} \mapsto_{\mathsf{c}} (\mathcal{C}[\overline{b}\ \langle \boldsymbol{y}\rangle] \parallel_a \mathcal{D}) \parallel_b \mathcal{D}[u^{b/\boldsymbol{y}}/a]$$

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

Table 4.2: Reduction rules for λ_{Cl}

with $\xi, \xi', \xi_1, \xi_2, \ldots$ the stacks of length 1. If t is a proof term, as usual $t\,\sigma$ denotes the term $(((t\,\xi_1)\,\xi_2)\ldots\xi_n)$.

We discuss now the last details of the reduction rules of λ_{Cl} in Table 4.2.

To trigger a cross reduction for $u \parallel_a v$ we require that the communication complexity of a is greater than 0. As this is a warning that the subformula property does not hold, we are using a logical property as a *computational criterion* for determining when a computation should start and stop. In λ-calculi the subformula property fares pretty well as a stopping criterion. In a sense, it detects all the *essential* operations that really have to be done. For example, in simply typed λ-calculus, a closed term that has the subformula property must be a *value*, that is, of the form $\lambda x\, u$, or $\langle u, v\rangle$. Indeed a closed term which is a not

a value, must be of the form $h\,\sigma$, for some stack σ (see Definition 4.1.10), where h is a redex $(\lambda y\,u)t$ or $\langle u, v\rangle\,\pi_i$; but $(\lambda y\,u)$ and $\langle u, v\rangle$ would have a more complex type than the type of the whole term, contradicting the subformula property. To see the subformula property at work for λ_{Cl}, consider again the term $\lambda y^B\,a^{\neg B}\,y \parallel_a x^{B\to\neg B}\,a^B : \neg B$ in the reduction 4.1. Since all prime factors of the communication kind B of a are proper subformulae of the type $\neg B$ of the term, the communication complexity of a is 0 and the cross reduction is not fired, thus avoiding the loop in 4.1.

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

Theorem 4.1.3 (Subject reduction)**.** *If $t : A$ and $t \mapsto_{\mathsf{c}} u$, then $u : A$ and all the free variables of u appear among those of t.*

Proof. See Theorem 2.4.3 for the case of intuitionistic reductions. We prove here the claim for cross reductions and for permutations since the case of simplification reductions is trivial and that of basic cross reductions is just a simpler version of that for cross reductions.

1. $(\ldots \parallel \mathcal{C}[\overline{a}\,v] \parallel \ldots) \parallel_a \mathcal{D} \mapsto_{\mathsf{c}} (\mathcal{C}[\overline{b}\,\langle \boldsymbol{y}\rangle] \parallel_a \mathcal{D}) \parallel_b \mathcal{D}[v^{\,b/\boldsymbol{y}}/a]$. Since $\langle \boldsymbol{y}\rangle : B$ and $\overline{b} : \neg B$, then $\overline{b}\,\langle \boldsymbol{y}\rangle : \bot$ and $\mathcal{C}[\overline{b}\,\langle \boldsymbol{y}\rangle]$ can be assigned the correct type. Since the types of $v^{\,b/\boldsymbol{y}}$ and a are the same, the term $\mathcal{D}[v^{\,b/\boldsymbol{y}}/a]$ is correctly defined. Finally, since $\overline{a}$ is rightmost in $\mathcal{C}[\overline{a}\,v]$ and $\boldsymbol{y}$ is the sequence of the free variables of t which are bound in $\mathcal{C}[\overline{a}\,v]$, by Definition 4.1.6, all free variables of $v^{\,b/\boldsymbol{y}}$ in $\mathcal{D}[v^{\,b/\boldsymbol{y}}/a]$ are also free in $\mathcal{C}[\overline{a}\,v]$. Hence, no new free variable is created during the reduction.

2. $(u \parallel_a v)w \mapsto uw \parallel_a vw$, if a does not occur free in w. All free variables of $(u \parallel_a v)w$ are also free variables of $uw \parallel_a vw$.

3. $w(u \parallel_a v) \mapsto wu \parallel_a wv$, if a does not occur free in w. All free variables of $wu \parallel_a wv$ are also free variables of $w(u \parallel_a v)$.

4. $w_1 \parallel_a w_2\,\mathsf{efq}_P \mapsto w_1\,\mathsf{efq}_P \parallel_a w_2\,\mathsf{efq}_P$, where P is atomic. The term $w_1\,\mathsf{efq}_P \parallel_a w_2\,\mathsf{efq}_P$ has the same free variables as $w_1 \parallel_a w_2\,\mathsf{efq}_P$.

5. $(u \parallel_a v)\pi_i \mapsto u\pi_i \parallel_a v\pi_i$, for $i = 0, 1$. The free variables of $(u \parallel_a v)\pi_i$ and of $u\pi_i \parallel_a v\pi_i$ are the same.

6. $\lambda x\,(u \parallel_a v) \mapsto \lambda x\,u \parallel_a \lambda x\,v$. The term $\lambda x\,(u \parallel_a v)$ and $\lambda x\,u \parallel_a \lambda x\,v$ have the same free variables.

7. $\langle u \parallel_a v,\, w\rangle \mapsto \langle u, w\rangle \parallel_a \langle v, w\rangle$, if a does not occur free in w. The free variables of the terms $\langle u \parallel_a v,\, w\rangle$ and $\langle u, w\rangle \parallel_a \langle v, w\rangle$ are the same.

8. $\langle w,\, u \parallel_a v\rangle \mapsto \langle w, u\rangle \parallel_a \langle w, v\rangle$, if a does not occur free in w.

   The case is symmetric to the previous one.

9. $(u \parallel_a v) \parallel_b w \mapsto (u \parallel_b w) \parallel_a (v \parallel_b w)$, if the communication complexity of b is greater than 0. The free variables of $(u \parallel_a v) \parallel_b w$ and $(u \parallel_b w) \parallel_a (v \parallel_b w)$ are the same.

10. $w \parallel_b (u \parallel_a v) \mapsto (w \parallel_b u) \parallel_a (w \parallel_b v)$, if the communication complexity of b is greater than 0. The free variables of the terms $w \parallel_b (u \parallel_a v)$ and $(w \parallel_b u) \parallel_a (w \parallel_b v)$ are the same.

11. $u \parallel_a v \mapsto u$, if a does not occur in u. The free variables of u are a subset of the free variables of $u \parallel_a v$.

12. $u \parallel_a v \mapsto v$, if a does not occur in v.

    This case is symmetric to the previous one.

□

Remark 4.1 (Non-determinism and non-confluence)**.** Since the simplification and communication reductions of λ_{Cl} are highly non-deterministic, the calculus λ_{Cl} is not confluent. Before triggering a communication, we need to choose a channel application to use. Although some constraints apply, we might still be able to choose among different channel applications and different choices might lead to different outcomes. Consider for example the term

$$t \quad = \quad ((\overline{a}\,(\lambda x^{A\wedge A\to A}\,\lambda y^{A\wedge A}\,y\pi_0))\,\mathsf{efq}_{A\wedge A\to A} \parallel (\overline{a}\,(\lambda z^{A\wedge A\to A} z))\,\mathsf{efq}_{A\wedge A\to A}) \;\parallel_a\; a\,(\lambda y^{A\wedge A}\,y\pi_1)$$

where $\overline{a} : \neg((A\wedge A\to A)\to(A\wedge A\to A))$ and $a : (A\wedge A\to A)\to(A\wedge A\to A)$. If we start by applying a basic cross reduction on the leftmost occurrence of $\overline{a}$, this term reduces as follows

$$t \quad \mapsto_{\mathsf{c}} \quad (\lambda x^{A\wedge A\to A}\,\lambda y^{A\wedge A}\,y\pi_0)(\lambda y^{A\wedge A}\,y\pi_1) \quad \mapsto_{\mathsf{c}} \quad \lambda y^{A\wedge A}\,y\pi_0$$

If we start instead applying a basic cross reduction to the rightmost occurrence of $\overline{a}$, the original term reduces as follows

$$t \quad \mapsto_{\mathsf{c}} \quad (\lambda z^{A\wedge A\to A} z)(\lambda y^{A\wedge A}\,y\pi_1) \quad \mapsto_{\mathsf{c}} \quad \lambda y^{A\wedge A}\,y\pi_1$$

From a logical viewpoint, both reductions yield a normal proof of the formula $A\wedge A\to A$, but the first one is obtained by the rule $\dfrac{A_0\wedge A_1}{A_0}$ and the second one by the rule $\dfrac{A_0\wedge A_1}{A_1}$.

Similarly, the choice of which term to eliminate by a simplification reduction might be relevant with respect to the result of the normalization. Consider for example the term

$$\begin{array}{rcl} s & = & (\lambda x^{\neg A\wedge B}\,\langle x^{\neg A\wedge B}\,\pi_1, x^{\neg A\wedge B}\,\pi_0\rangle) \;\parallel_a \\ & & \quad (\lambda y^{\neg A\wedge B}\,\langle y^{\neg A\wedge B}\,\pi_1, b^{\neg A}\rangle \;\parallel_b\; \lambda z^{\neg A\wedge B}\,(z^{\neg A\wedge B}\,\pi_0\,b^{A})\,\mathsf{efq}_{B\wedge\neg A}) \end{array}$$

We have two ways to apply a cimplification reduction to it:

$$s \quad \mapsto_{\mathsf{c}} \quad \lambda x^{\neg A\wedge B}\,\langle x^{\neg A\wedge B}\,\pi_1, x^{\neg A\wedge B}\,\pi_0\rangle$$

and

$$s \quad \mapsto_{\mathsf{c}} \quad \lambda y^{\neg A\wedge B}\,\langle y^{\neg A\wedge B}\,\pi_1, b^{\neg A}\rangle \;\parallel_b\; \lambda z^{\neg A\wedge B}\,(z^{\neg A\wedge B}\,\pi_0\,b^{A})\,\mathsf{efq}_{B\wedge\neg A}$$

Both applications are legitimate and the two results differ considerably. In the first case we obtain a simply typed λ-term corresponding to an intuitionistic proof of $\neg A\wedge B \to B\wedge\neg A$, while in the second we obtain a normal λ_{Cl} term corresponding to a classical proof of the same formula.

4.1.2 Subformula property

We start by defining the concepts of parallel form and normal form.

Definition 4.1.11 (Parallel form)**.** A *parallel form* is defined inductively as follows: a simply typed λ-term is a parallel form; if u and v are parallel forms, then both $u \parallel_a v$ and $u \parallel v$ are parallel forms.

Definition 4.1.12 (Normal forms)**.** We define NF_{c} to be the set of normal λ_{Cl}-terms as defined in Definition 2.4.2.

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

Proposition 4.1.4 (Bound hypothesis property)**.** *Suppose that $t \in \mathrm{NF}_{\mathsf{c}}$ is a simply typed λ-term, $x_1^{A_1}, \ldots, x_n^{A_n} \vdash t : A$ and $z : B$ is a variable occurring bound in t. Then one of the following holds: (1) B is a proper subformula of a prime factor of A or (2) B is a strong subformula of one among $A_1, \ldots, A_n$.*

Proof. By induction on t.

- $t = x_i^{A_i}$, with $1 \leq i \leq n$. Since by hypothesis z must occur bound in t, this case is impossible.
- $t = \lambda x^T u$, with $A = T \to U$. If $z = x^T$, since A is a prime factor of itself, we are done. If $z \neq x^T$, then z occurs bound in u and by induction hypothesis applied to $u : U$, we have two possibilities: i) B is a proper subformula of a prime factor of U and thus a proper subformula of a prime factor – A itself – of A; ii) B already satisfies 2., and we are done, or B is a strong subformula of T, and thus it satisfies 1.
- $t = \langle u_1, u_2\rangle$, with $A = T_1 \wedge T_2$. Then z occurs bound in u or v and, by induction hypothesis applied to $u_1 : T_1$ and $u_2 : T_2$, we have two possibilities: i) B is a proper subformula of a prime factor of T_1 or T_2, and thus B is a proper subformula of a prime factor of A as well; ii) B satisfies 2. and we are done.

- $t = u\,\mathsf{efq}_P$, with $A = P$. Then z occurs bound in u. Since $\bot$ has no proper subformula, by induction hypothesis applied to $u : \bot$, we have that B satisfies 2. and we are done.

- $t = x_i^{A_i}\,\xi_1 \ldots \xi_m$, where $m > 0$ and each ξ_j is either a term or a projection π_k. Since z occurs bound in t, it occurs bound in some term $\xi_j : T$, where T is a proper subformula of A_i. By induction hypothesis applied to ξ_j, we have two possibilities: i) B is a proper subformula of a prime factor of T and, by Definition 4.1.8, B is a strong subformula of A_i. ii) B satisfies 2. and we are done.

□

Each occurrence of a hypothesis in a normal intuitionistic proof is followed by an elimination rule, if the hypothesis is neither $\bot$ nor a subformula of the conclusion nor a proper subformula of some other premise.

Proposition 4.1.5 (The state of a variable)**.** *Let $t \in \mathrm{NF}_{\mathsf{c}}$ be a simply typed λ-term and $x_1^{A_1}, \ldots, x_n^{A_n}, z^B \vdash t : A$ One of the following holds:*

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

Proof. By induction on t.

- $t = x_i^{A_i}$, with $1 \leq i \leq n$. Trivial.

- $t = z^B$. This means that $B = A$, and we are done.

- $t = \lambda x^T u$, with $A = T \to U$. By induction hypothesis applied to $u : U$, we have two possibilities: i) every occurrence of z^B in u is of the form $z^B\,\xi$, and we are done; ii) $B = \bot$ or B is a subformula of U, and hence of A, or a proper subformula of one among the formulae $A_1, \ldots, A_n$, and we are done again.

- $t = \langle u_1, u_2 \rangle$, with $A = T_1 \wedge T_2$. By induction hypothesis applied to $u_1 : T_1$ and $u_2 : T_2$, we have two possibilities: i) every occurrence of z^B in u_1 and u_2 is of the form $z^B\,\xi$, and we are done; ii) $B = \bot$ or B is a subformula of T_1 or T_2, and hence of A, or a proper subformula of one among the formulae $A_1, \ldots, A_n$, and we are done again.

- $t = u\,\mathsf{efq}_P$, with $A = P$. By induction hypothesis applied to $u : \bot$, we have two possibilities: i) every occurrence of z^B in u is of the form $z^B\,\xi$, and we are done; ii) $B = \bot$ or a proper subformula of one among $A_1, \ldots, A_n$, and we are done again.

- $t = x_i^{A_i}\,\xi_1 \dots \xi_m$, where $m > 0$ and each ξ_j is either a term or a projection π_k. Suppose there is an i such that in the term $\xi_j : T_j$ not every occurrence of z^B in u is of the form $z^B\,\xi$. If $B = \bot$, we are done. If not, then by induction hypothesis B is a subformula of T_j or a proper subformula of one among $A_1, \dots, A_n$. Since T_j is a proper subformula of A_i, in both cases B is a proper subformula of one among $A_1, \dots, A_n$.

- $t = z^B\,\xi_1 \dots \xi_m$, where $m > 0$ and each ξ_i is either a term or a projection π_j. Suppose there is an i such that in the term $\xi_i : T_i$ not every occurrence of z^B in u is of the form $z^B\,\xi$. If $B = \bot$, we are done. If not, then by induction hypothesis B is a subformula of T_i or a proper subformula of one among $A_1, \dots, A_n$. But the former case is not possible, since T_i is a proper subformula of B, hence the latter holds.

□

We prove now that the subformula property holds for normal λ_{Cl}-terms. This intuitively means that the proofs do not contain concepts which are not already contained in their assumptions or conclusion and that the programs do not use sub-procedures whose type is more complex than the type of their inputs and output.

The following statement has two parts, a stronger statement for communication variables, and a slightly weaker one for other terms. The reason why we can use a stronger notion of subformula property for communication variables is that, if we want the weaker version to hold for generic subterms, we need to simplify the type of communication variables even more.

Theorem 4.1.6 (Subformula property)**.** *Suppose that* $x_1^{A_1}, \dots, x_n^{A_n} \vdash t : A$ *and* $t \in \mathrm{NF}_{\mathsf{c}}$. *Then:*

1. *For each communication variable* a *occurring bound in* t *and with communication kind* B*, the prime factors of* B *are proper subformulae of* $A_1, \dots, A_n, A$.

2. *For each subterm* t *of type* B *that is not a bound communication variable,* B *is either a subformula or a conjunction of subformulae of* $A_1, \dots, A_n, A$.

Proof. By Proposition 4.1.7 if we remove the subscripts $t = t_1 \parallel t_2 \parallel \dots \parallel t_p$ and each t_i, for $1 \leq i \leq p$, is a simply typed λ-term.

By induction on t.

- $t = x_i^{A_i}$, with $1 \leq i \leq n$. Trivial.

- $t = \lambda x^T u$, with $A = T \to U$. By Proposition 4.1.7, t is a simply typed λ-term, so t contains no bound communication variable. Moreover, by induction hypothesis applied to $u : U$, the type of any subterm of u which is not a bound communication variable is either a subformula or a conjunction of subformulae of the formulae $T, A_1, \ldots, A_n, U$ and hence of the formulae $A_1, \ldots, A_n, A$.

- $t = \langle u_1, u_2 \rangle$, with $A = T_1 \wedge T_2$. By Proposition 4.1.7, t is a simply typed λ-term, so t contains no bound communication variable. Moreover, by induction hypothesis applied to $u_1 : T_1$ and $u_2 : T_2$, the type of any subterm of u which is not a bound communication variable is either a subformula or a conjunction of subformulae of the formulae $A_1, \ldots, A_n, T_1, T_2$ and hence of the formulae $A_1, \ldots, A_n, A$.

- $t = u_1 \parallel_b u_2$. Let C be the communication kind of b, we first show that the communication complexity of b is 0. We reason by contradiction and assume that it is greater than 0. u_1, u_2 are either simple parallel terms or of the form $v_1 \parallel_c v_2$. The second case is not possible, otherwise a permutation reduction could be applied to $t \in \mathrm{NF}_{\mathtt{c}}$. Thus u_1, u_2 are simple parallel terms. Since the communication complexity of b is greater than 0, the types of the occurrences of b in u_1, u_2 are not subformulae of $A_1, \ldots, A_n, A$. By Proposition 4.1.5, all occurrences of b in u_1, u_2 are of the form $\bar{b}\, w$ for some term w. Hence, we can write

  $$t = (\ldots \parallel \mathcal{C}[\bar{b}\ t] \parallel \ldots) \parallel_b \mathcal{D}$$

  where $\mathcal{C}[\ ]$ is a simple context, $(\ldots \parallel \mathcal{C}[\bar{b}\ t] \parallel \ldots)$ is a normal simple parallel term, $\mathcal{D}$ is a normal term, and b is rightmost in $\mathcal{C}[\bar{b}\ t]$. Hence a cross reduction of t can be performed, which contradicts the fact that $t \in \mathrm{NF}_{\mathtt{c}}$. Since we have established that the communication complexity of b is 0, the prime factors of C must be proper subformulae of $A_1, \ldots, A_n, A$. Now, by induction hypothesis applied to $u_1 : A, u_2 : A$, for each communication variable a^F occurring bound in t, the prime factors of F are proper subformulae of the formulae $A_1, \ldots, A_n, A$ or subformulae of C and thus of the formulae $A_1, \ldots, A_n, A$; moreover, the type of any subterm of u_1, u_2 which is not a communication variable is either a subformula or a conjunction of subformulae of the formulae $A_1, \ldots, A_n, C$ and thus of $A_1, \ldots, A_n, A$.

- $t = x_i^{A_i}\, \xi_1 \ldots \xi_m$, where $m > 0$ and each $\xi_j : T_j$ is either a term or a projection π_k and T_j is a subformula of A_i. By Proposition 4.1.7, t is a simply typed λ-term, so t contains no bound communication variable. By induction hypothesis applied to each $\xi_j : T_j$, the type of any subterm of t which is not a bound communication variable is either a subformula or a conjunction of subformulae of the formulae $A_1, \ldots, A_n, T_1, \ldots, T_m$ and thus of the formulae $A_1, \ldots, A_n, A$.

- $t = u\, \mathsf{efq}_P$, with $A = P$. Then $u = x_i^{A_i}\, \xi_1 \ldots \xi_m$, where $m > 0$ and each ξ_j is either a term or a projection π_k. Hence, $\bot$ is a subformula of A_i. Finally, by the proof of the previous case, we obtain the thesis for t.

□

4.1.3 The normalization theorem

We prove that every proof term of λ_{Cl} reduces in a finite number of steps to a normal form. By subject reduction this implies the normalization for $\mathrm{N}\mathfrak{I}^{\rightarrow}_{\wedge\bot} + (\mathrm{EM})$ proofs.

We shall define a reduction strategy for terms of λ_{Cl}: a recipe for selecting, in any given term, the subterm to which apply one of our basic reductions. We remark that the permutations between communications have been adopted to simplify the normalization proof, but at the same time, they undermine strong normalization, because they enable silly loops, like in cut-elimination for sequent calculi.

The idea behind our normalization strategy is to employ a suitable complexity measure for terms $u \parallel_a v$ and, each time a reduction has to be performed, to choose the term of maximal complexity. Since cross reductions can be applied as long as there is a violation of the subformula property, the natural approach is to define the complexity measure as a function of some fixed set of formulae, representing the formulae that can be safely used without violating the subformula property.

Proposition 4.1.7 (Parallel form property)**.** *If $t \in \mathrm{NF}_{\mathsf{c}}$ is a λ_{Cl}-term, then it is in parallel 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 $u_1 \parallel_a u_2$, otherwise one could apply the permutation

  $$t = \lambda x^A\, u_1 \parallel_a u_2 \mapsto_{\mathsf{c}} \lambda x^A\, u_1 \parallel_a \lambda x^A\, u_2$$

  and t would not be in normal form. Hence, by induction hypothesis v must be a simply typed λ-term.
- $t = \langle v_1, v_2 \rangle$. Since t is normal, neither v_1 nor v_2 can be of the form $u_1 \parallel_a u_2$, otherwise one could apply one of the permutations

  $$\langle u_1 \parallel_a u_2,\, w \rangle \mapsto_{\mathsf{c}} \langle u_1, w \rangle \parallel_a \langle u_2, w \rangle$$

  $$\langle w,\, u_1 \parallel_a u_2 \rangle \mapsto_{\mathsf{c}} \langle w, u_1 \rangle \parallel_a \langle w, u_2 \rangle$$

  and t would not be in normal form. Hence, by induction hypothesis v_1 and v_2 must be simply typed λ-terms.
- $t = v_1\, v_2$. Since t is normal, neither v_1 nor v_2 can be of the form $u_1 \parallel_a u_2$, otherwise one could apply one of the permutations

  $$w\,(u_1 \parallel_a u_2) \mapsto_{\mathsf{c}} wu_1 \parallel_a wu_2$$

$$(u_1 \parallel_a u_2)\, w \mapsto_{\mathsf{c}} u_1 w \parallel_a u_2 w$$

and t would not be in normal form. Hence, by induction hypothesis v_1 and v_2 must be simply typed λ-terms.

- $t = v\,\mathsf{efq}_P$. Since t is normal, v cannot be of the form $u_1 \parallel_a u_2$, otherwise one could apply the permutation

$$u_1 \parallel_a u_2\,\mathsf{efq}_P \mapsto_{\mathsf{c}} u_1\,\mathsf{efq}_P \parallel_a u_2\,\mathsf{efq}_P$$

and t would not be in normal form. Hence, by induction hypothesis u_1 and u_2 must be simply typed λ-terms.

- $t = u\,\pi_i$. Since t is normal, v cannot be of the form $u_1 \parallel_a u_2$, otherwise one could apply the permutation

$$(u_1 \parallel_a u_2)\,\pi_i \mapsto_{\mathsf{c}} u_1\pi_i \parallel_a u_2\pi_i$$

and t would not be in normal form. Hence, by induction hypothesis u must be a simply typed λ-term, which is the thesis.

- $t = u_1 \parallel_a u_2$. By induction hypothesis the thesis holds for u_i where $1 \leq i \leq 2$ and hence trivially for t.

□

Definition 4.1.13 (Complexity of parallel terms)**.** Let $\mathcal{A}$ be a finite set of formulae. The $\mathcal{A}$-*complexity* of $u \parallel_a v$ is the sequence (c, d, l, o) of natural numbers, where:

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

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

The following normalization algorithm represents the constructive content of the proofs of Proposition 4.1.8 and Theorem 4.1.10. Essentially, the master reduction strategy consists in iterating the basic reduction relation $\succ_{\mathsf{c}}$ defined in Definition 4.1.14 below, whose goal is to permute the smallest redex $u \parallel_a v$ of maximal complexity until u, v are simple parallel terms, see Definition 4.1.3, then normalize them and apply cross reductions.

Definition 4.1.14 (Side reduction strategy)**.** Let $t : A$ be a term with free variables $x_1^{A_1}, \ldots, x_n^{A_n}$ and $\mathcal{A}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \ldots, A_n$. Let $u \parallel_a v$ be the *smallest subterm* of t, if any, among those of *maximal* $\mathcal{A}$-complexity and let (c, d, l, o) be its $\mathcal{A}$-complexity. We write $t \succ_{\mathsf{c}} t'$ whenever t' has been obtained from t by applying to $u \parallel_a v$:

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

Definition 4.1.15 (Master reduction strategy)**.** We define a normalization algorithm $\mathcal{N}(t)$ taking as input a typed term t and producing a term t' such that $t \mapsto_{\mathsf{c}}^* t'$. Assume that the free variables of t are $x_1^{A_1}, \ldots, x_n^{A_n}$ and let $\mathcal{A}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \ldots, A_n$. The algorithm behaves as follows.

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

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

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

Proposition 4.1.8. *Let $t : A$ be any term. Then $t \mapsto_{\mathsf{c}}^* t'$, where t' is a parallel form.*

Proof. By induction on t. For the present proof, we write $u \rightrightarrows^* u''$ to indicate that $u \mapsto_{\mathsf{c}} u'$ and u' can be denoted as u'' omitting parentheses and the subscript of $\parallel$ operators.

- t is a variable x. Trivial.
- $t = \lambda x\, u$. By induction hypothesis,

  $$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

  and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying $n-1$ times the permutations we obtain

  $$t \rightrightarrows^* \lambda x\, u_1 \parallel \lambda x\, u_2 \parallel \ldots \parallel \lambda x\, u_n$$

  which is the thesis.
- $t = u\, v$. By induction hypothesis,

  $$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

  $$v \rightrightarrows^* v_1 \parallel v_2 \parallel \ldots \parallel v_m$$

  and each term u_i and v_i, for $1 \leq i \leq n, m$, is a simply typed λ-term. Applying $(n-1)+(m-1)$ times the permutations we obtain

  $$\begin{aligned} t &\rightrightarrows^* (u_1 \parallel u_2 \parallel \ldots \parallel u_n)\, v \\ &\rightrightarrows^* u_1\, v \parallel u_2\, v \parallel \ldots \parallel u_n\, v \\ &\rightrightarrows^* u_1\, v_1 \parallel u_1\, v_2 \parallel \ldots \parallel u_1\, v_m \parallel \ldots \\ &\qquad \ldots \parallel u_n\, v_1 \parallel u_n\, v_2 \parallel \ldots \parallel u_n\, v_m \end{aligned}$$

- $t = \langle u, v \rangle$. By induction hypothesis,

  $$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

  $$v \rightrightarrows^* v_1 \parallel v_2 \parallel \ldots \parallel v_m$$

  and each term u_i and v_i, for $1 \leq i \leq n, m$, is a simply typed λ-term. Applying $(n-1)+(m-1)$ times the permutations we obtain

  $$\begin{aligned} t &\rightrightarrows^* \langle u_1 \parallel u_2 \parallel \ldots \parallel u_n,\, v \rangle \\ &\rightrightarrows^* \langle u_1, v \rangle \parallel \langle u_2, v \rangle \parallel \ldots \parallel \langle u_n, v \rangle \\ &\rightrightarrows^* \langle u_1, v_1 \rangle \parallel \langle u_1, v_2 \rangle \parallel \ldots \parallel \langle u_1, v_m \rangle \parallel \ldots \\ &\qquad \ldots \parallel \langle u_n, v_1 \rangle \parallel \langle u_n, v_2 \rangle \parallel \ldots \\ &\qquad \ldots \parallel \langle u_n, v_m \rangle \end{aligned}$$

- $t = u\,\pi_i$. By induction hypothesis,

$$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying $n-1$ times the permutations we obtain

$$t \rightrightarrows^* u_1\,\pi_i \parallel u_2\,\pi_i \parallel \ldots \parallel u_n\,\pi_i.$$

- $t = u\,\mathsf{efq}_P$. By induction hypothesis,

$$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying $n-1$ times the permutations we obtain

$$t \rightrightarrows^* u_1\,\mathsf{efq}_P \parallel u_2\,\mathsf{efq}_P \parallel \ldots \parallel u_n\,\mathsf{efq}_P$$

□

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

Lemma 4.1.9. *Let $t : A$ be a term in parallel form which is not simply typed and. Let $\mathcal{A}$ contain all proper subformulae of A and be closed under subformulae. Assume that $r > 0$ is the maximum $\mathcal{A}$-communication-complexity of the subterms of t. Assume that the free variables $x_1^{A_1}, \ldots, x_n^{A_n}$ of t are such that for every i, either each strong subformula of A_i is in $\mathcal{A}$, or each proper prime subformula of A_i is in $\mathcal{A}$ or has at most r symbols. Suppose moreover that no subterm $u_1 \parallel_a u_2$ with $\mathcal{A}$-communication-complexity r contains a subterm of the same $\mathcal{A}$-communication-complexity. Then there exists t' such that $t \succ_{\mathsf{c}}^* t'$ and the maximal among the $\mathcal{A}$-communication-complexity of the subterms of t' is strictly smaller than r.*

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

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

(a) $\rho = (r, d, l, o)$, with $d > 0$. We first show that the term $u_1 \parallel_a u_2$ is a redex. Now, the free variables of $u_1 \parallel_a u_2$ are among $x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{B_1}, \ldots, a_p^{B_p}$ and the communication kind of a is D. Hence, suppose by contradiction that all the prime factors of D are proper subformulae of A or strong subformulae of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$. Given that $r > 0$ there is a prime factor P of D such that P has r symbols and does not belong to $\mathcal{A}$. The possible cases are two: (i) P is a proper subformula of a prime proper subformula A_i' of A_i such that $A_i' \notin \mathcal{A}$; (ii) P, by Proposition 4.1.2, is a proper subformula of a prime factor of B_i. If (i), then the number of symbols of A_i' is less than

or equal to r, so P cannot be a proper subfomula of A'_i, which is a contradiction. If (ii), then, since by hypothesis $a_i^{B_i}$ is bound in t, there is a prime factor of B_i having a number of symbols greater than r, hence we conclude that there is a subterm $w_1 \parallel_b w_2$ of t having $\mathcal{A}$-complexity greater than ρ, which is absurd.

Now, since $d > 0$, we may assume that for some $1 \leq i \leq 2$, $u_i = w_1 \parallel_b w_2$. Suppose $i = 2$. The term $u_1 \parallel_a (w_1 \parallel_b w_2)$ is then a redex of t and by replacing it with $(*)$: $(u_1 \parallel_a w_1) \parallel_b (u_1 \parallel_a w_2)$ we obtain from t a term t' such that $t \succ_{\mathsf{c}} t'$ according to Definition 4.1.14. We must verify that we can apply to t' the main induction hypothesis. Indeed, the reduction $t \succ_{\mathsf{c}} t'$ duplicates all the subterms of u_1, but all of their $\mathcal{A}$-complexities are smaller than r, because $u_1 \parallel_a u_2$ by choice is the smallest subterm of t having maximal $\mathcal{A}$-complexity ρ. The terms $(u_1 \parallel_a w_i)$ for $1 \leq i \leq 2$ have smaller $\mathcal{A}$-complexity than ρ, because they have numbers of occurrences of the symbol $\parallel$ strictly smaller than in $u_1 \parallel_a u_2$. Moreover, the terms in t' with $(*)$ as a subterm have, by hypothesis, $\mathcal{A}$-communication-complexity smaller than r and hence $\mathcal{A}$-complexity smaller than ρ. Assuming that the communication kind of b is F, the prime factors of F that are not in $\mathcal{A}$ must have fewer symbols than the prime factors of D that are not in $\mathcal{A}$, again because $u_1 \parallel_a u_2$ by choice is the smallest subterm of t having maximal $\mathcal{A}$-complexity ρ; hence, the $\mathcal{A}$-complexity of $(*)$ is smaller than ρ. Hence the number of subterms of t' with $\mathcal{A}$-complexity ρ is strictly smaller than k. By induction hypothesis, $t' \succ_{\mathsf{c}}^* t''$, where t'' satisfies the thesis.

(b) $\rho = (r, d, l, o)$, with $d = 0$ and $l > 0$. Since $d = 0$, u_1, u_2 are simple parallel terms – and thus strongly normalizable [GLT89] – so we may assume that for $1 \leq i \leq 2$, $u_i \mapsto_{\mathsf{c}}^* u'_i \in \mathrm{NF}_{\mathsf{c}}$ by a sequence intuitionistic reduction rules. By replacing in t the subterm $u_1 \parallel_a u_2$ with $u'_1 \parallel_a u'_2$, we obtain a term t' such that $t \succ_{\mathsf{c}} t'$ according to Definition 4.1.14. Moreover, the terms in t' with $u'_1 \parallel_a u'_2$ as a subterm have, by hypothesis, $\mathcal{A}$-communication-complexity smaller than r and hence $\mathcal{A}$-complexity is smaller than ρ. By induction hypothesis, $t' \succ_{\mathsf{c}}^* t''$, where t'' satisfies the thesis.

(c) $\rho = (r, d, l, o)$, with $d = l = 0$. Since $d = 0$, u_1, u_2 are simply typed λ-terms. Since $l = 0$, u_1, u_2 are in normal form and thus satisfy conditions 1. and 2. of Proposition 4.1.4. We need to check that $u_1 \parallel_a u_2$ is a redex, in particular that the communication complexity of a is greater than 0. Assume that the free variables of $u_1 \parallel_a u_2$ are among $x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{B_1}, \ldots, a_p^{B_p}$ and that the communication kind of a is D. As we argued above, we obtain that not all the prime factors of D are proper subformulae of A or strong subformulae of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$. By Definition 4.1.9, $u_1 \parallel_a u_2$ is a redex.

We now prove that every occurrence of a in u_1, u_2 is of the form $a\,\xi$ for some term or projection ξ. First of all, a occurs with arrow type in all u_1, u_2. Moreover, $u_1 : A, u_2 : A$, since $t : A$ and t is a parallel form; hence, the types of the occurrences of a in u_1, u_2 cannot be subformulae of A, otherwise $r = 0$, and cannot be proper subformulae of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$, otherwise the prime factors of D would be strong subformulae of one among $A_1, \ldots, A_n, B_1, \ldots, B_p$ and thus we are done. Thus by Proposition 4.1.5 we

are done. Two cases can occur.

• a does not occur in u_i for $1 \leq i \leq 2$. By performing a simplification reduction, we replace in t the term $u_1 \parallel_a u_2$ with u_i and obtain a term t' such that $t \succ_{\mathsf{c}} t'$ according to Definition 4.1.14. After the replacement, the number of subterms having maximal $\mathcal{A}$-complexity ρ in t' is strictly smaller than the number of such subterms in t. By induction hypothesis, $t' \succ_{\mathsf{c}}^* t''$, where t'' satisfies the thesis.

• a occurs in all the subterms u_1, u_2. Let $u_1 = (\ldots \parallel \mathcal{C}[\overline{a}\ w] \parallel \ldots)$ where $\overline{a} : \neg D$, $(\ldots \parallel \mathcal{C}[\overline{a}\ w] \parallel \ldots)$ is a normal simple parallel term, $\mathcal{C}[\,]$ is a simple context, and the displayed occurrence of $\overline{a}$ is rightmost in $\mathcal{C}[\overline{a}\ w]$. By applying a cross reduction to $\mathcal{C}[\overline{a}\ w] \parallel_a u_2$ we obtain either the term $u_2[w/a]$ or the term $(**)$ $(\mathcal{C}[\overline{b}\,\langle \boldsymbol{y} \rangle] \parallel_a u_2) \parallel_b u_2[w^{\,b/\boldsymbol{y}}/a]$ where $\overline{b} : \neg B$, $\boldsymbol{y}$ is the sequence of the free variables of w_z which are bound in $\mathcal{C}[\overline{a}\ w]$ and $\overline{a}$ does not occur in w. In the former case, the term has $\mathcal{A}$-complexity strictly smaller than ρ and we are done. In the latter case, since u_1, u_2 satisfy conditions 1. and 2. of Proposition 4.1.4, the types $Y_1, \ldots, Y_k$ of the variables $\boldsymbol{y}$ are proper subformulae of A or strong subformulae of the formulae $A_1, \ldots, A_n, B_1, \ldots, B_p$. Hence, the types among $Y_1, \ldots, Y_k$ which are not in $\mathcal{A}$ are strictly smaller than all the prime factors of the formulae $B_1, \ldots, B_p$. Since the communication kind of b consists of the formulae $Y_1 \wedge \ldots \wedge Y_k$, by Definition 4.1.13 the $\mathcal{A}$-complexity of the term $(**)$ above is strictly smaller than the $\mathcal{A}$-complexity ρ of $u_1 \parallel_a u_2$.

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

(d) $\rho = (r, d, l, o)$, with $d = l = o = 0$. Since $o = 0$, $u_1 \parallel_a u_2$ is a redex. Let us say a does not occur in u_i for $1 \leq i \leq 2$. By performing a simplification reduction, we replace $u_1 \parallel_a u_2$ with u_i according to Definition 4.1.14. Hence, by induction hypothesis, $t' \succ_{\mathsf{c}}^* t''$, where t'' satisfies the thesis. □

Theorem 4.1.10. *Let $t : A$ be a λ_{Cl}-term. Then $t \mapsto_{\mathsf{c}}^* t' : A$, where t' is a normal parallel form.*

Proof. By Proposition 4.1.8, we can assume that $t : A$ is in parallel form. Assume now that the free variables of t are $x_1^{A_1}, \ldots, x_n^{A_n}$ and let $\mathcal{A}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \ldots, A_n$. We prove the theorem by lexicographic induction on the quadruple $(|\mathcal{A}|, r, k, s)$ where $|\mathcal{A}|$ is the cardinality of $\mathcal{A}$, r is the maximal $\mathcal{A}$-communication-complexity of the subterms of t, k is the number of subterms of t having maximal $\mathcal{A}$-communication-complexity r and s is the size of t. If t

is a simply typed λ-term, it has a normal form [GLT89] and we are done; so we assume t is not. There are two main cases.

• *First case*: *t is not a redex.* Let $t = u_1 \parallel_a u_2$ and let C be the communication kind of a. Then, the communication complexity of a is 0 and by Definition 4.1.9 every prime factor of C belongs to $\mathcal{A}$. Let the types of the occurrences of a in u_i for $1 \leq i \leq 2$ be B_i, with $B_i = \neg C$ or $B_i = C$. Let now $\mathcal{A}_i$ be the set of the proper subformulae of A and the strong subformulae of $A_1, \ldots, A_n, B_i$. By Proposition 4.1.2, every strong subformula of B_i is a proper subformula of a prime factor of C, and this prime factor is in $\mathcal{A}$. Hence, $\mathcal{A}_i \subseteq \mathcal{A}$.

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

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

Let now $t' = u_1' \parallel_a u_2'$, so that $t \mapsto_{\mathsf{c}}^* t'$. If t' is normal, we are done. Otherwise, since u_j' for $1 \leq j \leq 2$ are normal, the only possible redex remaining in t' is the whole term itself, i.e., $u_1' \parallel_a u_2'$: this happens only if the free variables of t' are fewer than those of t; w.l.o.g., assume they are $x_1^{A_1}, \ldots, x_i^{A_i}$, with $i < n$. Let $\mathcal{B}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \ldots, A_i$. Since t' is a redex, the communication complexity of a is greater than 0; by Definition 4.1.9, a prime factor of C is not in $\mathcal{B}$, so we have $\mathcal{B} \subset \mathcal{A}$. By I.H., $t' \mapsto_{\mathsf{c}}^* t''$, where t'' is a parallel normal form.

• *Second case*: *t is a redex.* Let $u_1 \parallel_a u_2$ be the *smallest* subterm of t having $\mathcal{A}$-communication-complexity r. The free variables of $u_1 \parallel_a u_2$ satisfy the hypotheses of Lemma 4.1.9 either because have type A_i and $\mathcal{A}$ contains all the strong subformulae of A_i, or because the prime proper subformulae of their type have at most r symbols, by maximality of r. By Lemma 4.1.9 $u_1 \parallel_a u_2 \succ_{\mathsf{c}}^* w$ where the maximal among the $\mathcal{A}$-communication-complexity of the subterms of w is strictly smaller than r. Let t' be the term obtained replacing w for $u_1 \parallel_a u_2$ in t. We apply the I.H. and obtain $t' \mapsto_{\mathsf{c}}^* t''$ with t'' in parallel normal form. □

4.1.4 On the expressive power of λ_{Cl}

We discuss now the expressive power of λ_{Cl} and its computational capabilities. First, we establish the relation of λ_{Cl} with the simply typed λ-calculus and Parigot's λ_μ [Par92].

Proposition 4.1.11. *λ_{Cl} is strictly more expressive than simply typed λ-calculus. Moreover, λ_{Cl} cannot be simulated by propositional λ_μ-calculus.*

Proof. The simply typed λ-calculus can be trivially embedded into λ_{Cl}. The converse does not hold, as λ_{Cl} can encode the parallel OR, which is a term $\mathsf{O} : \mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool}$ such that $\mathsf{O}\mathtt{ff}\mathtt{ff} \mapsto_{\mathsf{c}}^* \mathtt{ff}$, $\mathsf{O}u\mathtt{tt} \mapsto_{\mathsf{c}}^* \mathtt{tt}$, $\mathsf{O}\mathtt{tt}u \mapsto_{\mathsf{c}}^* \mathtt{tt}$ for every term u. A consequence of

Berry's sequentiality theorem [Bar84], is that there is no parallel OR in simply typed λ-calculus. O can instead be defined in Boudol's parallel λ-calculus [Bou89]. Assuming to add the boolean type in our calculus, that $\overline{a} : \mathtt{Bool} \wedge S \to \bot$, and that $a : \mathtt{Bool} \wedge S$, "$\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$" is as usual, the λ_{Cl}-term for such parallel OR is

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

for any flag term $s : S$, introduced to control the communication complexity of a and make it reducible by communications in all circumstances. Now, $\mathsf{O}\,u\,\mathtt{tt}$ reduces to $\mathtt{tt}$ by

$$(\mathsf{if}\ u\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ \overline{a}\,\langle \mathtt{ff}, s\rangle\, \mathsf{efq}_{\mathtt{Bool}}) \parallel_a (\mathsf{if}\ \mathtt{tt}\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ a\,\pi_0)$$

$$\mapsto_{\mathsf{c}}^* (\mathsf{if}\ u\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ \overline{a}\,\langle \mathtt{ff}, s\rangle\, \mathsf{efq}_{\mathtt{Bool}}) \parallel_a \mathtt{tt} \ \mapsto_{\mathsf{c}}\ \mathtt{tt}$$

And symmetrically $\mathsf{O}\,\mathtt{tt}\,u \mapsto_{\mathsf{c}}^* \mathtt{tt}$. On the other hand, $\mathsf{O}\,\mathtt{ff}\,\mathtt{ff}$ reduces to $\mathtt{ff}$ by

$$(\mathsf{if}\ \mathtt{ff}\ \mathsf{then}\ \mathtt{ff}\ \mathsf{else}\ \overline{a}\,\langle \mathtt{ff}, s\rangle\, \mathsf{efq}_{\mathtt{Bool}}) \parallel_a (\mathsf{if}\ \mathtt{ff}\ \mathsf{then}\ \mathtt{ff}\ \mathsf{else}\ a\,\pi_0)$$

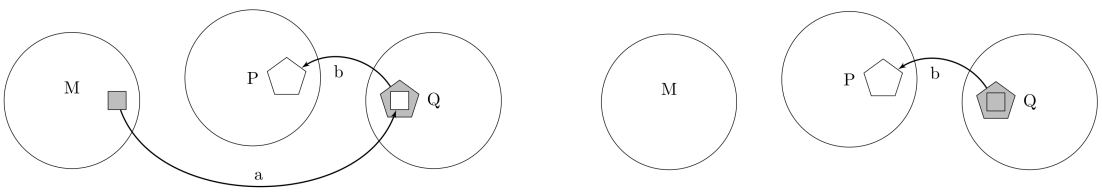
$$\mapsto_{\mathsf{c}}^* \overline{a}\,\langle \mathtt{ff}, s\rangle\, \mathsf{efq}_{\mathtt{Bool}} \parallel_a a\,\pi_0 \quad \mapsto_{\mathsf{c}} \quad \langle \mathtt{ff}, s\rangle\,\pi_0 \quad \mapsto_{\mathsf{c}} \quad \mathtt{ff}$$

The claim about λ_μ follows by Ong's embedding of propositional λ_μ in the simply typed λ-calculus, see Lemma 6.3.7 of [SU98]. Indeed the translation $\underline{u}$ of a λ_μ-term u is such that $\underline{s\,t} = \underline{s}\,\underline{t}$ and $\underline{x} = x$ for any variable x, if there were a typed λ_μ-term O for parallel OR, then

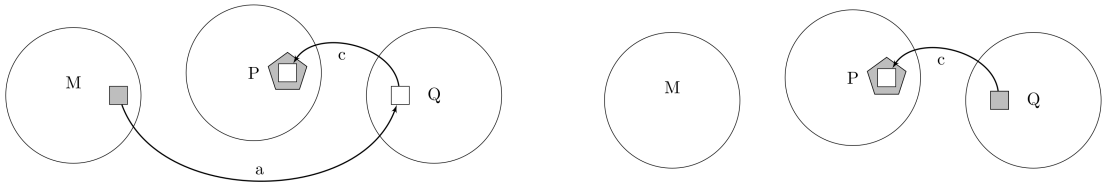
$$\begin{array}{rcccl}
\underline{\mathsf{O}}\,x\,\mathtt{tt} & = & \underline{\mathsf{O}\,x\,\mathtt{tt}} & \mapsto_{\mathsf{c}}^* & \underline{\mathtt{tt}} \ = \ \mathtt{tt} \\
\underline{\mathsf{O}}\,\mathtt{tt}\,x & = & \underline{\mathsf{O}\,\mathtt{tt}\,x} & \mapsto_{\mathsf{c}}^* & \underline{\mathtt{tt}} \ = \ \mathtt{tt} \\
\underline{\mathsf{O}}\,\mathtt{ff}\,\mathtt{ff} & = & \underline{\mathsf{O}\,\mathtt{ff}\,\mathtt{ff}} & \mapsto_{\mathsf{c}}^* & \underline{\mathtt{ff}} \ = \ \mathtt{ff}
\end{array}$$

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

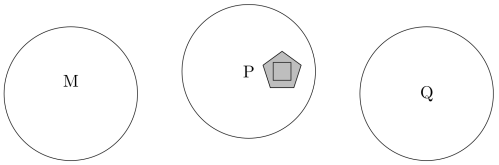
Example 4.1.3 (**Cross reductions for program efficiency**)**.** We show how to use cross reductions to communicate processes that are still waiting for some arguments. Consider the process $M \parallel_a (Q \parallel_b P)$. The process Q contains a channel b to send a message (gray pentagon) to P (below left), but the message is missing a part (gray square) which is computed by M and sent to Q by a. In a system without a closure handling mechanism, the whole interaction needs to wait until M can communicate to Q (below right).



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



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

Both reductions terminate then with 

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

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

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

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

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

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

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

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

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

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

4.1.5 Classical disjunction

We present now the formal definition of disjunction and disjunction rules in N$\mathfrak{Cl}$ and we define the corresponding computational constructs in λ_{Cl}.

Definition 4.1.16 (Definition of $\vee$ in **CL**)**.** For any formula A, we define the translation $A^{\vee c}$ inductively as follows:

- for any propositional variable p, $p^{\vee c} = p$
- $\bot^{\vee c} = \bot$
- $(B \to C)^{\vee c} = B^{\vee c} \to C^{\vee c}$
- $(B \wedge C)^{\vee c} = B^{\vee c} \wedge C^{\vee c}$
- $(B \vee C)^{\vee c} = (B^{\vee c} \to \bot) \to (C^{\vee c} \to \bot) \to \bot$

For any set of formulae Γ, we denote by $\Gamma^{\vee c}$ the set $\{A^{\vee c} : A \in \Gamma\}$.

We show now the completeness of N$\mathfrak{Cl}$ with respect to the $^{\vee c}$ translation of the theorems of classical logic.

Proposition 4.1.12. *For any set Γ of formulae and formula A, if $\Gamma \vdash_{\mathbf{CL}} A$, then $\Gamma^{\vee c} \vdash_{\mathrm{N}\mathfrak{Cl}} A^{\vee c}$.*

Proof. By Theorem 4.1.1, there is a derivation $\mathcal{D}$ of A in N$\mathfrak{I}$ + (EM). The proof is by induction on the height l of $\mathcal{D}$.

If $\mathcal{D}$ has height 0, then it is A and our N$\mathfrak{Cl}$ proof is just $A^{\vee c}$.

We reason now by cases on the last rule applied in $\mathcal{D}$:

- $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ C\end{array}}{B \to C}$$

  with assumptions Γ. By induction hypothesis there exists a derivation

  $$\begin{array}{c}\mathcal{E}\\ C^{\vee c}\end{array}$$

  in N$\mathfrak{Cl}$ with assumptions $\Gamma^{\vee c}$ and possibly $B^{\vee c}$. We construct the derivation

  $$\dfrac{\begin{array}{c}\mathcal{E}\\ C^{\vee c}\end{array}}{B^{\vee c} \to C^{\vee c}}$$

possibly discharging $B^{\vee c}$ and obtain the required derivation of $A^{\vee c}$ in N$\mathfrak{Cl}$ since $(B \to C)^{\vee c} = B^{\vee c} \to C^{\vee c}$.

- $$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2 \\ B \to C & B\end{matrix}}{C}$$

  By induction hypothesis there exist derivations

  $$\begin{matrix}\mathcal{E}_1 & & \mathcal{E}_2 \\ (B \to C)^{\vee c} & & B^{\vee c}\end{matrix}$$

  in N$\mathfrak{Cl}$ whose assumptions are contained in $\Gamma^{\vee c}$. Since $(B \to C)^{\vee c} = B^{\vee c} \to C^{\vee c}$ the required derivation is

  $$\dfrac{\begin{matrix}\mathcal{E}_1 & \mathcal{E}_2 \\ B^{\vee c} \to C^{\vee c} & B^{\vee c}\end{matrix}}{C^{\vee c}}$$

- $$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2 \\ B & C\end{matrix}}{B \wedge C}$$

  By induction hypothesis there exist derivations

  $$\begin{matrix}\mathcal{E}_1 & & \mathcal{E}_2 \\ B^{\vee c} & & C^{\vee c}\end{matrix}$$

  in N$\mathfrak{Cl}$ whose assumptions are contained in $\Gamma^{\vee c}$. Since $(B \wedge C)^{\vee c} = B^{\vee c} \wedge C^{\vee c}$ the required derivation is

  $$\dfrac{\begin{matrix}\mathcal{E}_1 & \mathcal{E}_2 \\ B^{\vee c} & C^{\vee c}\end{matrix}}{B^{\vee c} \wedge C^{\vee c}}$$

- $$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D} \\ B_1 \wedge B_2\end{matrix}}{B_i}$$

  By induction hypothesis there exists a derivation

  $$\begin{matrix}\mathcal{E} \\ (B_1 \wedge B_2)^{\vee c}\end{matrix}$$

  in N$\mathfrak{Cl}$ whose assumptions are $\Gamma^{\vee c}$. Since $(B_1 \wedge B_2)^{\vee c} = B_1^{\vee c} \wedge B_2^{\vee c}$ the required derivation is

  $$\dfrac{\begin{matrix}\mathcal{E} \\ B_1^{\vee c} \wedge B_2^{\vee c}\end{matrix}}{B_i^{\vee c}}$$

- $$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D} \\ \bot\end{matrix}}{P}$$

for some atomic formula $P \neq \bot$. By induction hypothesis there exists a derivation

$$\begin{array}{c}\mathcal{E}\\ \bot^{\vee c}\end{array}$$

in N$\mathfrak{CI}$ whose assumptions are $\Gamma^{\vee c}$. Since $\bot^{\vee c} = \bot$ and $P^{\vee c} = P$ the required derivation is

$$\dfrac{\begin{array}{c}\mathcal{E}\\ \bot^{\vee c}\end{array}}{P^{\vee c}}$$

- $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}\\ B_i\end{array}}{B_1 \vee B_2}$$

  By induction hypothesis there exists a derivation

  $$\begin{array}{c}\mathcal{E}\\ B_i^{\vee c}\end{array}$$

  in N$\mathfrak{CI}$ whose assumptions are $\Gamma^{\vee c}$. Since $(B_1 \vee B_2)^{\vee c} = (B_1^{\vee c} \to \bot) \to (B_2^{\vee c} \to \bot) \to \bot$ the required derivation is

  $$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{E}\\ B_i^{\vee c}\end{array} \quad [B_i^{\vee c} \to \bot]}{\bot}\,2}{(B_2^{\vee c} \to \bot) \to \bot}}{(B_1^{\vee c} \to \bot) \to (B_2^{\vee c} \to \bot) \to \bot}\,1$$

  where $B_i^{\vee c} \to \bot$ is discharged by 1 if $i = 1$ and by 2 of $i = 2$.

- $$\mathcal{D} = \dfrac{\begin{array}{ccc}\mathcal{D}_0 & \mathcal{D}_1 & \mathcal{D}_2\\ B_1 \vee B_2 & C & C\end{array}}{C}$$

  By induction hypothesis there exist derivations

  $$\begin{array}{ccc}\begin{array}{c}\mathcal{E}_0\\ (B_1 \vee B_2)^{\vee c}\end{array} & \qquad \begin{array}{c}\mathcal{E}_1\\ C^{\vee c}\end{array} & \qquad \begin{array}{c}\mathcal{E}_2\\ C^{\vee c}\end{array}\end{array}$$

  in N$\mathfrak{CI}$ where the assumptions of $\mathcal{E}_0$ are included in $\Gamma^{\vee c}$, and the assumptions of $\mathcal{E}_i$ for $i \in \{1, 2\}$ are included in $\Gamma^{\vee c}$ plus possibly $B_i^{\vee c}$. Since $(B_1 \vee B_2)^{\vee c} = (B_1^{\vee c} \to \bot) \to (B_2^{\vee c} \to \bot) \to \bot$ the required derivation is

  $$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{E}_0\\ (B_1^{\vee c} \to \bot) \to (B_2^{\vee c} \to \bot) \to \bot\end{array} \quad [\neg B_1^{\vee c}]^1}{(B_2^{\vee c} \to \bot) \to \bot} \quad [\neg B_2^{\vee c}]^2}{\dfrac{\bot}{C^{\vee c}}}}{C^{\vee c}}\,(\mathrm{EM})^2 \qquad \dfrac{\begin{array}{c}\mathcal{E}_1\\ C^{\vee c}\end{array}}{C^{\vee c}}\,(\mathrm{EM})^1$$

where each $(\mathrm{EM})^i$ discharges the displayed occurrences of $\neg B_i^{\vee c}$ and the occurrences of $B_i^{\vee c}$ in $\mathcal{E}_i$. By Proposition 2.4.2, we know that the derivation of $C^{\vee c}$ from $\bot$ denoted as $\dfrac{\bot}{\dfrac{}{C^{\vee c}}}$ always exists. Such derivation has exactly the assumptions of $\mathcal{E}_0$.

□

The natural deduction rules for disjunction correspond to the computational constructs of case distinction and injection. Since disjunction can be defined in $\mathrm{N}\mathfrak{Cl}$, we can define this constructs in λ_{Cl}.

For the sake of simplicity we denote here by $w\,\mathsf{efq}_T$, for any term $w : \bot$ and complex type T, the term $s\,w$ where $s : \bot \to T$ is constructed λ-abstracting the term obtained from the derivation in Proposition 2.4.2.

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

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

where $a : A$, $\overline{a} : A \to \bot$, $b : B$,$\overline{b} : B \to \bot$, $v_0 : F$, $v_1 : F$, $t : A \vee B$. We can then verify, for example, that

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

$$\mapsto_{\mathsf{c}}^* ((\overline{a}\;u)\,\mathsf{efq}_F \parallel_a v_0[a/x_0])) \parallel_b v_1[b/x_1] \mapsto_{\mathsf{c}} v_0[u/x_0] \parallel_b v_1[b/x_1] \mapsto_{\mathsf{c}} v_0[u/x_0]$$

4.2 Adding the symmetry: λ_{G}

There is more than one way to transmit a message through a channel. We present λ_{G}, a typed concurrent extension of λ-calculus, the channels of which establish a symmetric connection between two parallel processes. While many logical symmetries are proper to classical logic, the axiom used to type communication channels in λ_{Cl} imposes a very precise direction to the communication: from the variable $\overline{a} : \neg A$ to the variable $a : A$. There is no reduction that enables $a : A$ to transmit to $\overline{a} : \neg A$, only reductions enabling it to receive from $\overline{a} : \neg A$. The logical reason for this is that an argument of $\overline{a} : \neg A$ has to be a term of type A, which is precisely the kind of term that can replace an occurrence of $a : A$; while an argument of $a : A$ cannot, in general, replace an application of $\overline{a} : \neg A$.

If we want a single channel endpoint to be able, in general, both to send and receive messages without type violations, we need an axiom schema that identifies the type of the argument of each communication channel with the type of the output of the other channel. The simplest solution to the equation is $\mathrm{Lin} = (A \to B) \vee (B \to A)$: any argument of $a^{A\to B}$ can be used to replace the result of an application of $a^{B\to A}$, and

vice versa. Extending N$\mathfrak{I}$ by the axiom Lin, we obtain the natural deduction N$\mathfrak{G}$ for Gödel–Dummett logic **GL**.

Hence the parallelism operator of λ_{G} implements a symmetric channel connecting two processes:

Figure 4.2: Representation of a λ_{G} channel.

The calculus N$\mathfrak{G}$ is of interest also from a purely proof-theoretical point of view. Indeed, it solves the problem of finding an analytic natural deduction for **GL** in a novel and quite simple way. Other attempts in this direction [BCF00, BP15] are discussed in Section 2.4.3.

4.2.1 Natural deduction

From a logical point of view, the rule

$$\dfrac{\begin{array}{cc}[A \to B] & [B \to A] \\ \vdots & \vdots \\ C & C\end{array}}{C}\ (\mathrm{Lin})$$

is equivalent to the higher-level rule

$$\dfrac{\begin{array}{cc}\dfrac{A}{B} & \dfrac{B}{A} \\ \vdots & \vdots \\ C & C\end{array}}{C}$$

defined in Section 3.3.2. We choose this version of the rule for the reasons already discussed in Section 4.1.1 with regard to the rule (EM) of λ_{Cl}. The soundness and completeness of L$\mathfrak{I}$ extended by this rule with respect to **GL** follows from Theorem 3.3.1. Nonetheless, a direct proof that L$\mathfrak{I}$ + (Lin) is sound and complete with respect to **GL** follows.

Theorem 4.2.1 (Soundness and completeness)**.** *For any set* Π *of formulae and formula* A, $\Pi \vdash_{\mathrm{N}\mathfrak{I}+(\mathrm{Lin})} A$ *if and only if* $\Pi \vdash_{\mathbf{GL}} A$.

Proof. The calculus N$\mathfrak{I}$ is strongly equivalent to any Hilbert calculus for intuitionistic logic – see for example Chapter 2 of [TS96]. Moreover, we can capture **GL** extending such systems by the axiom schema Lin

Hence for the left to right direction, we show that we can simulate any instance of the rule (Lin) using instances of the Lin axiom in N$\mathfrak{I}$ as follows:

$$\dfrac{(A \to B) \vee (B \to A) \qquad \begin{array}{c}[A \to B]^1 \\ \vdots \\ C\end{array} \qquad \begin{array}{c}[B \to A]^1 \\ \vdots \\ C\end{array}}{C}\ {\scriptstyle 1}$$

While for the right to left direction, we show that we can derive any instance of the Lin axiom as follows:

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

□

We exploit the definability of $A \vee B$ as $((B \to A) \to A) \wedge ((A \to B) \to B)$ in **GL**, see [Dum59], and treat $\vee$ as a defined connective. We provide all details in Definition 4.2.9 and Proposition 4.2.9 in Section 4.2.7.

Definition 4.2.1 (Natural deduction calculus N$\mathfrak{G}$)**.** The natural deduction calculus N$\mathfrak{G}$ extends N$\mathfrak{I}^{\to}_{\wedge\bot}$ by the (Lin) rule.

4.2.2 Reduction rules

We present now the reduction rules for N$\mathfrak{G}$. As usual, a normal deduction in N$\mathfrak{G}$ should have two essential features: every intuitionistic Prawitz-style reduction should have been carried out and the subformula property should hold. Due to the (Lin) rule, the former is not always enough to guarantee the latter. Here we present the main ideas behind the normalization procedure for N$\mathfrak{G}$ and the needed reduction rules. The computational interpretation of the rules will be carried out through the λ_{G} calculus in Section 4.2.3.

The normalization procedure is similar to the procedure for λ_{Cl}. The mains steps are the following:

- We permute down all applications of (Lin).

Obtained a deduction in *parallel form*, we interleave the following two steps.

- We apply the standard intuitionistic reductions [Pra71] to the parallel branches of the derivation.

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

- We apply *cross reductions* to replace the (Lin) applications that violate the subformula property.

Using these reductions, we eliminate the detours that appear in configurations like the one below on the left. To remove them, a tentative idea could be to simultaneously move the deduction $\mathcal{D}_1$ to the right and $\mathcal{D}_2$ to the left thus obtaining the derivation below right:

$$\dfrac{\dfrac{\mathcal{D}_1}{A}\quad[A\to B]}{\begin{array}{c}B\\ \vdots\\ C\end{array}}\qquad\dfrac{\dfrac{\mathcal{D}_2}{B}\quad[B\to A]}{\begin{array}{c}A\\ \vdots\\ C\end{array}}\qquad\qquad\dfrac{\begin{array}{c}\mathcal{D}_2\\ B\\ \vdots\\ C\end{array}\qquad\begin{array}{c}\mathcal{D}_1\\ A\\ \vdots\\ C\end{array}}{C}$$

but the unrestricted transformation above cannot work; indeed $\mathcal{D}_1$ might contain the hypothesis $A \to B$ and hence it cannot be moved on the right. Even worse, $\mathcal{D}_1$ may depend on hypotheses that are locally opened, but discharged below B but above C. Again, it is not possible to move $\mathcal{D}_1$ on the right as naively thought, otherwise new global hypotheses would be created.

As in the case of λ_{Cl}, cross reductions solve this problem. Let us highlight Γ and Δ, the hypotheses of $\mathcal{D}_1$ and $\mathcal{D}_2$ that are respectively discharged below B and A but above the application of (Lin). Assume moreover, that $A \to B$ does not occur in $\mathcal{D}_1$ and $B \to A$ does not occur in $\mathcal{D}_2$ as hypotheses discharged by (Lin). A cross reduction transforms the deduction below left into the deduction below right if (Lin) in the original proof discharges in each branch exactly one occurrence of the hypotheses, and Γ and Δ are formulae

$$\dfrac{\begin{array}{c}\Gamma\\ \mathcal{D}_1\\ A\end{array}\quad[A\to B]}{\begin{array}{c}B\\ \vdots\\ C\end{array}}\qquad\dfrac{\begin{array}{c}\Delta\\ \mathcal{D}_2\\ B\end{array}\quad[B\to A]}{\begin{array}{c}A\\ \vdots\\ C\end{array}}\qquad\qquad\dfrac{\dfrac{\Delta}{\Gamma}\quad[\Delta\to\Gamma]}{\begin{array}{c}\mathcal{D}_1\\ A\\ \vdots\\ C\end{array}}\qquad\dfrac{\dfrac{\Gamma}{\Delta}\quad[\Gamma\to\Delta]}{\begin{array}{c}\mathcal{D}_2\\ B\\ \vdots\\ C\end{array}}$$

and into the following deduction, in the general case

$$\dfrac{\dfrac{\dfrac{\Gamma}{\mathcal{D}_1}}{\dfrac{A \quad [A \to B]}{B}}_1 \quad \dfrac{\dfrac{\dfrac{\Delta}{\Gamma}}{\mathcal{D}_1}}{A}_3}{\dfrac{C \qquad\qquad C}{C}_1} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma}{\Delta}}{\mathcal{D}_2}}{B}_3 \quad \dfrac{\dfrac{\Delta}{\mathcal{D}_2}}{\dfrac{B \quad [B \to A]}{A}_2}}{\dfrac{C \qquad\qquad C}{C}_2}}{C}_3$$

where the notation $\dfrac{X}{\dfrac{}{Y}}$ means that we derive each occurrence of element D of Y as follows:

$$\dfrac{\dfrac{\dfrac{X}{}}{\bigwedge X}\,\wedge i \qquad [\bigwedge X \to \bigwedge Y]}{\dfrac{\bigwedge Y}{D}\,\wedge e}$$

where $\bigwedge X$ and $\bigwedge Y$ are the conjunctions of the elements of X and Y respectively, $\dfrac{\dfrac{X}{}}{\bigwedge X}\,\wedge i$ is the obvious derivation by conjunction introductions, $\bigwedge X \to \bigwedge Y$ is discharged by (Lin) and $\dfrac{\bigwedge Y}{D}\,\wedge e$ is the obvious derivation of D by conjunction eliminations.

Mindless applications of the cross reductions might produce reduction loops, see for instance Example 4.2.1. To avoid them we will allow cross reductions to be performed only when the proof is not analytic.

4.2.3 The λ_{G}-calculus

We introduce λ_{G}, a parallel λ-calculus for **GL**. λ_{G} extends simply typed λ-calculus with a parallel operator that interprets the inference for the linearity axiom Lin. We describe λ_{G}-terms and their computational behavior, proving as main result of the section the subject reduction theorem, stating that the reduction rules preserve the type.

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

Table 4.3: Type assignment for λ_{G}.

Definition 4.2.2 (Terms of λ_{G})**.** The terms of λ_{G} are defined by the rules for simply typed λ-calculus in Table 2.5 and by the rules in Table 4.3.

The calculus λ_{G} is isomorphic to N$\mathfrak{G}$. Parallelism is introduced by the rule for the linearity axiom Lin.

Proof terms may contain variables $x_0^A, x_1^A, x_2^A, \ldots$ of type A for every formula A. ; these variables are denoted as x^A, y^A, $z^A, \ldots$, $a^A, b^A, c^A, \ldots$ For clarity, the variables introduced by the (Lin) rule will be often denoted with letters $a, b, c, \ldots$ but they are not in a syntactic category apart. A variable x^A that occurs in a term of the form $\lambda x^A u$ is called λ-*variable* and a variable a that occurs in a term $u \parallel_a v$ is called *communication variable* and represents a private communication channel between the parallel processes u and v.

The free and bound variables of a proof term are defined as usual and for the new term $u \parallel_a v$, all the free occurrences of a in u and v are bound in $u \parallel_a v$.

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

In order to determine how complex the communication channel a of a term $u \parallel_a v$ is, we use logic as we did for λ_{Cl}. First, we consider the types B, C such that a occurs with type $B \to C$ in u and thus with type $C \to B$ in v. Then, assume that $u \parallel_a v$ has type A and its free variables are $x_1^{A_1}, \ldots, x_n^{A_n}$. The subformula property tells us that, when the computation is over, the type of any object will not be more complex than the types of the inputs and of the output. Hence, if the prime factors of the types B and C are not subformulae of $A_1, \ldots, A_n, A$, then these prime factors should be taken into account in the complexity measure we are looking for. The relative definition is the following.

Definition 4.2.3 (Communication complexity)**.** Let $u \parallel_a v : A$ a proof term with free variables $x_1^{A_1}, \ldots, x_n^{A_n}$. Assume that $a^{B \to C}$ occurs in u and thus $a^{C \to B}$ in v.

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

We explain now the basic reduction rules for the proof terms of λ_{G}, which are given in Table 4.4.

Basic cross reductions As in λ_{Cl} these reductions can be triggered only when the free variables of the transmitted term u are free in the original environment $\mathcal{C}[a\, u]$. As usual, basic cross reductions implement a simple transmission of data or executable code, but the calculus λ_{G} features two versions of this reduction because the message can be communicated from left to right:

$$\mathcal{C}[a^{A \to B}\, u] \parallel_a \mathcal{D}[a^{B \to A}\, v] \;\mapsto_{\mathtt{g}}\; \mathcal{C}[a^{A \to B}\, u] \parallel_a \mathcal{D}[u]$$

or from right to left:

$$\mathcal{D}[a^{A\to B}\, v] \parallel_a \mathcal{C}[a^{B\to A}\, u] \;\mapsto_{\mathtt{g}}\; \mathcal{C}[u] \parallel_a \mathcal{C}[a^{B\to A}\, u]$$

Cross reductions As in λ_{Cl}, these reductions address the problem of transmitting function closures, see [EBPJ11]. The solution provided by λ_{G} as well is that function closures are transmitted in two steps: first, the function code, then, when it is available, the evaluation environment. As a result, we can communicate open λ_{G}-terms which are closed in their original environment. The process of handling and transmitting function closures is typed by a new instance of (Lin).

The conditions of a cross reduction are equivalent to those employed for λ_{Cl} and are similarly motivated. Notice that also λ_{G} requires a termination criterion for the computation because, as shown in Example 4.2.1, unrestricted cross reductions do not always terminate.

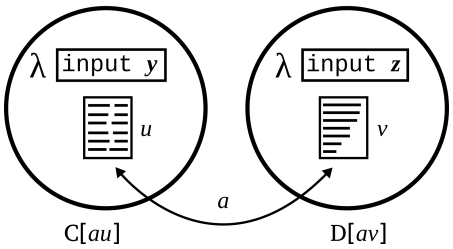
Let us consider now the reduction itself:

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

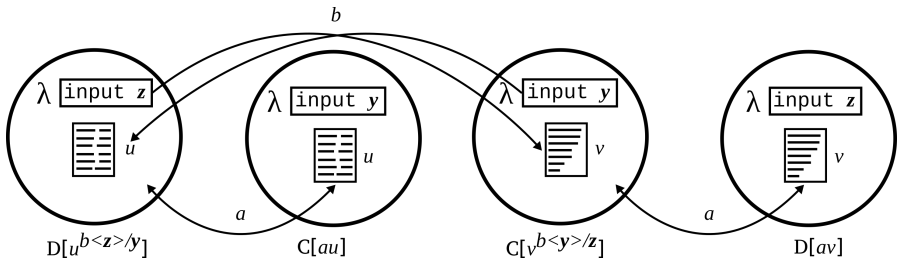
where $\boldsymbol{y}$ is the sequence of the free variables of u which are bound in $\mathcal{C}[a\, u]$, $\boldsymbol{z}$ is the sequence of the free variables of v which are bound in $\mathcal{D}[a\, v]$, and C and D are the conjunctions of the types of the variables in $\boldsymbol{z}$ and $\boldsymbol{y}$, respectively.

Unlike in λ_{Cl} cross reductions, the communication here is symmetric: the term u is transmitted to the right in order to replace $a\, v$ and the term v is transmitted to the left in order to replace $a\, u$. The free variables $\boldsymbol{y}$ of u which are bound in $\mathcal{C}[a\, u]$ and the variables $\boldsymbol{z}$ of v which are bound by $\mathcal{D}[a\, v]$ are handled in a symmetric way by the new communication channel b.

If we represent the configuration of the term before the reduction as follows:



After the reduction we obtain the term



In which b reconnects the messages u and v to their original environments by the substitutions $b\langle \boldsymbol{z}\rangle/\boldsymbol{y}$ and $b\langle \boldsymbol{y}\rangle/\boldsymbol{z}$ respectively. Thus, when available, $\boldsymbol{y}$ will be sent to u and $\boldsymbol{z}$ to v through the channel b. Note that in the result of the cross reduction the processes $\mathcal{C}[a\,u]$ and $\mathcal{D}[a\,v]$ are *cloned*, because their code fragments can be needed again.

Example 4.2.1. Let y and z be bound variables occurring in the normal terms $\mathcal{C}[a\,y]$ and $\mathcal{D}[a\,z]$. Without the condition on the communication complexity c of a, a loop could be generated:

$$\mathcal{C}[a\,y] \parallel_a \mathcal{D}[a\,z] \mapsto_{\mathsf{g}} (\mathcal{D}[y^{b\langle z\rangle/y}] \parallel_a \mathcal{C}[a\,y]) \parallel_b (\mathcal{C}[z^{b\langle y\rangle/z}] \parallel_a \mathcal{D}[a\,z])$$

$$= (\mathcal{D}[b\,z] \parallel_a \mathcal{C}[a\,y]) \parallel_b (\mathcal{C}[b\,y] \parallel_a \mathcal{D}[a\,z]) \mapsto_{\mathsf{g}}^* \mathcal{D}[b\,z] \parallel_b \mathcal{C}[b\,y]$$

In Section 4.2.5 we show that no loop occurs if $c > 0$.

Permutation reductions Just like those of λ_{Cl}, these reductions regulate the interaction between parallel operators and the other computational constructs. The permutations between the parallel operators and the other operators are used to obtain a parallel form, see Proposition 4.2.3, which is completely analogous to the parallel form of λ_{Cl}.

Everything works as expected: the reductions steps in Table 4.4 preserve the type at the level of proof terms, namely they correspond to logically sound proof transformations.

Theorem 4.2.2 (Subject reduction)**.** *If $t : A$ and $t \mapsto_{\mathsf{g}} u$, then $u : A$ and all the free variables of u appear among those of t.*

Proof. See Theorem 2.4.3 and 4.1.3 for intuitionistic reductions and permutations. Simplification reductions just require a trivial argument and basic cross reductions a simplified version of the argument for cross reductions. As for cross reductions, suppose that

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

Since $\langle \boldsymbol{y}\rangle : C := C_0 \wedge \ldots \wedge C_n$ and $\langle \boldsymbol{z}\rangle : D := D_0 \wedge \ldots \wedge D_m$, $b^{D\to C}\langle \boldsymbol{z}\rangle$ and $b^{C\to D}\langle \boldsymbol{y}\rangle$ are correct terms. Therefore $u^{b^{D\to C}\langle \boldsymbol{z}\rangle/\boldsymbol{y}}$ and $v^{b^{C\to D}\langle \boldsymbol{y}\rangle/\boldsymbol{z}}$, by Definition 4.1.6, are correct as well. The assumptions are that $\boldsymbol{y} = y_0^{C_0}, \ldots, y_n^{C_n}$ is the sequence of the free variables of u which are bound in $\mathcal{C}[a^{A\to B}u]$, $\boldsymbol{z} = z_0^{D_0}, \ldots, z_m^{D_m}$ is the sequence of the free variables

Intuitionistic reductions

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

Parallel Operator Permutations

$$w(u \parallel_a v) \mapsto_{\mathsf{g}} wu \parallel_a wv \text{ if } a \text{ does not occur free in } w$$
$$(u \parallel_a v)\xi \mapsto_{\mathsf{g}} u\xi \parallel_a v\xi \text{ if } \xi \text{ is a one-element stack and } a \text{ does not occur free in } \xi$$
$$w(u \parallel v) \mapsto_{\mathsf{g}} wu \parallel wv \qquad (u \parallel v)\xi \mapsto_{\mathsf{g}} u\xi \parallel v\xi \qquad \lambda x^A\,(u \parallel_a v) \mapsto_{\mathsf{g}} \lambda x^A\, u \parallel_a \lambda x^A\, v$$
$$\langle u \parallel_a v,\, w\rangle \mapsto_{\mathsf{g}} \langle u, w\rangle \parallel_a \langle v, w\rangle \qquad \langle w,\, u \parallel_a v\rangle \mapsto_{\mathsf{g}} \langle w, u\rangle \parallel_a \langle w, v\rangle \qquad \lambda x^A\,(u \parallel v) \mapsto_{\mathsf{g}} \lambda x^A\, u \parallel \lambda x^A\, v$$
$$\langle u \parallel v,\, w\rangle \mapsto_{\mathsf{g}} \langle u, w\rangle \parallel \langle v, w\rangle \qquad \langle w,\, u \parallel v\rangle \mapsto_{\mathsf{g}} \langle w, u\rangle \parallel \langle w, v\rangle$$
$$(u \parallel_a v) \parallel_b w \mapsto_{\mathsf{g}} (u \parallel_b w) \parallel_a (v \parallel_b w)$$
if the communication complexity of b is greater than 0
$$w \parallel_b (u \parallel_a v) \mapsto_{\mathsf{g}} (w \parallel_b u) \parallel_a (w \parallel_b v)$$
if the communication complexity of b is greater than 0
$$(u \parallel v) \parallel_b w \mapsto_{\mathsf{g}} (u \parallel_b w) \parallel (v \parallel_b w)$$
if the communication complexity of b is greater than 0
$$w \parallel_b (u \parallel v) \mapsto_{\mathsf{g}} (w \parallel_b u) \parallel (w \parallel_b v)$$
if the communication complexity of b is greater than 0

Communication reductions

Basic cross reductions
$$\mathcal{C}[a^{A\to B}\, u] \parallel_a \mathcal{D}[a^{B\to A}\, v] \;\mapsto_{\mathsf{g}}\; \mathcal{C}[a^{A\to B}\, u] \parallel_a \mathcal{D}[u]$$
$$\mathcal{D}[a^{A\to B}\, v] \parallel_a \mathcal{C}[a^{B\to A}\, u] \;\mapsto_{\mathsf{g}}\; \mathcal{C}[u] \parallel_a \mathcal{C}[a^{B\to A}\, u]$$
where $\mathcal{C}[a\, u], \mathcal{D}[a\, v]$ are normal simply typed λ-terms and $\mathcal{C}, \mathcal{D}$ simple contexts; the free variables of u are also free in $\mathcal{C}[a\, u]$; the displayed occurrences of a are the rightmost both in $\mathcal{C}[a\, u]$ and in $\mathcal{D}[a\, v]$; and the communication complexity of a is greater than 0

Simplification reductions
$$u \parallel_a v \mapsto_{\mathsf{g}} u \quad \text{if } a \text{ does not occur in } u$$
$$u \parallel_a v \mapsto_{\mathsf{g}} v \quad \text{if } a \text{ does not occur in } v$$

Cross reductions

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

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

Table 4.4: Reduction Rules for λ_{G}

of v which are bound in $\mathcal{D}[a^{B\to A}v]$, a does not occur neither in u nor in v and b is fresh. Therefore, by construction all the variables $\boldsymbol{z}$ are bound in $\mathcal{D}[u^{b^{D\to C}\langle \boldsymbol{z}\rangle/\boldsymbol{y}}]$ and all the variables $\boldsymbol{y}$ are bound in $\mathcal{C}[v^{b^{C\to D}\langle \boldsymbol{y}\rangle/\boldsymbol{z}}]$. Hence, no new free variable is created. □

Definition 4.2.4 (Normal forms)**.** We define NF_{g} to be the set of normal λ_{G}-terms as defined in Definition 2.4.2.

Definition 4.2.5 (Parallel form)**.** A *parallel form* is defined inductively as follows: a simply typed λ-term is a parallel form; if u and v are parallel forms, then both $u \parallel_a v$ and $u \parallel v$ are parallel forms.

4.2.4 The subformula property

We show that normal λ_{G}-terms satisfy the subformula property (Theorem 4.2.4). This, in turn, implies that our Curry–Howard correspondence for λ_{G} is meaningful from the logical perspective and produces analytic N$\mathfrak{G}$ proofs.

Proposition 4.2.3 (Parallel normal form property)**.** *If $t \in \mathrm{NF}_{\mathtt{g}}$ is a λ_{G}-term, then it is in parallel form.*

Proof. Easy structural induction on t using the permutation reductions. See 4.1.7. □

We prove the subformula property for normal λ_{G}-terms.

Theorem 4.2.4 (Subformula property)**.** *Suppose*

$$x_1^{A_1}, \ldots, x_n^{A_n} \vdash t : A \quad \textit{and} \quad t \in \mathrm{NF}_{\mathtt{g}}. \quad \textit{Then}:$$

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

Proof. We proceed by induction on t. By Proposition 4.2.3 $t = t_1 \parallel_{a_1} t_2 \parallel_{a_2} \ldots \parallel_{a_n} t_{n+1}$ and each t_i, for $1 \leq i \leq n+1$, is a simply typed λ-term. We only show the case $t = u_1 \parallel_b u_2$ since the rest is already shown in the proof of Theorem 4.1.6. Let C, D be the communication kind of b, we first show that the communication complexity of b is 0. We reason by contradiction and assume that it is greater than 0. u_1 and u_2 are either simply typed λ-terms or of the form $v \parallel_c w$. The second case is not possible, otherwise a permutation reduction could be applied to $t \in \mathrm{NF}_{\mathtt{g}}$. Thus u_1 and u_2 are simply typed λ-terms. Since the communication complexity of b is greater than 0, the types $C \to D$ and $D \to C$ are not subformulae of $A_1, \ldots, A_n, A$. By Proposition 4.1.5, every occurrence of $b^{C \to D}$ in u_1 is of the form $b^{C \to D} v$ and every occurrence of $b^{D \to C}$ in u_2 is of the form $b^{D \to C} w$. Hence, we can write

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

where $\mathcal{C}, \mathcal{D}$ are simple contexts and b is rightmost. Hence a cross reduction of t can be performed, which contradicts the fact that $t \in \mathrm{NF}_{\mathtt{g}}$. Since we have established that the communication complexity of b is 0, the prime factors of C and D must be proper subformulae of $A_1, \ldots, A_n, A$. Now, by induction hypothesis applied to $u_1 : A$ and $u_2 : A$,

for each communication variable $a^{F\to G}$ occurring bound in t, the prime factors of F and G are proper subformulae of the formulae $A_1,\dots,A_n,A,C\to D,D\to C$ and thus of the formulae $A_1,\dots,A_n,A$; moreover, the type of any subterm of u_1 or u_2 which is not a communication variable is either a subformula or a conjunction of subformulae of the formulae $A_1,\dots,A_n,C\to D,D\to C$ and thus of $A_1,\dots,A_n,A$. □

4.2.5 The normalization theorem

Our goal is to prove the normalization theorem for λ_{G}: every proof term of λ_{G} reduces in a finite number of steps to a normal form. By subject reduction, this implies that N𝔊 proofs normalize.

The normalization strategy and the measures employed in the following normalization proof are comparable to those employed in the normalization proof for λ_{Cl}, with due distinction. Nonetheless, the cross reduction rules for the two calculi present substantial differences and hence require specific arguments, see for example point (c) of Lemma 4.2.6.

The complexity measure that we employ for $u\parallel_a v$ λ_{G}-terms is similar to that employed for λ_{Cl} in Section 4.1.3 and the proof is very similar as well.

Definition 4.2.6 (Complexity of parallel terms)**.** Let $\mathcal{A}$ be a finite set of formulae. The $\mathcal{A}$-*complexity* of the term $u\parallel_a v$ is the sequence (c,d,l,o) of natural numbers, where:

1. if the communication kind of a is B,C, then c is the maximum among 0 and the number of symbols of the prime factors of B or C that are not subformulae of some formula in $\mathcal{A}$;
2. d is the number of occurrences of $\parallel$ in u and v;
3. l is the sum of the lengths of the intuitionistic reductions of u and v to reach intuitionistic normal form;
4. o is the number of occurrences of a in u and v.

As for λ_{Cl}, the master reduction strategy consists in iterating the basic reduction relation $\succ_{\mathsf{g}}$ defined below, whose goal is to permute the smallest redex $u\parallel_a v$ of maximal complexity until u and v are simply typed λ-terms, then normalize them and finally apply the cross reductions.

Definition 4.2.7 (Side reduction strategy)**.** Let $t:A$ be a term with free variables $x_1^{A_1},\dots,x_n^{A_n}$ and $\mathcal{A}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1,\dots,A_n$. Let $u\parallel_a v$ the *smallest subterm* of t, if any, among those of *maximal* $\mathcal{A}$-complexity and let (c,d,l,o) its $\mathcal{A}$-complexity. We write

$$t\succ_{\mathsf{g}} t'$$

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

1. a permutation reduction

$$(u_1 \parallel_b u_2) \parallel_a v \mapsto_{\mathsf{g}} (u_1 \parallel_a v) \parallel_b (u_2 \parallel_a v)$$

$$u \parallel_a (v_1 \parallel_b v_2) \mapsto_{\mathsf{g}} (u \parallel_a v_1) \parallel_b (u \parallel_a v_2)$$

   if $d > 0$ and $u = u_1 \parallel_b u_2$ or $v = v_1 \parallel_b v_2$;

2. a sequence of intuitionistic reductions normalizing both u and v, if $d = 0$ and $l > 0$;

3. a cross reduction, or basic cross reduction, if $d = l = 0$ and $c > 0$, immediately followed by intuitionistic reductions normalizing the newly generated simply typed λ-terms and, if possible, by applications of the simplification reductions $u_1 \parallel_b v_1 \mapsto_{\mathsf{g}} u_1$ and $u_1 \parallel_b v_1 \mapsto_{\mathsf{g}} v_1$ to the whole term.

4. a simplification reduction $u \parallel_a v \mapsto_{\mathsf{g}} u$ and $u \parallel_a v \mapsto_{\mathsf{g}} v$ if $d = l = c = 0$.

Definition 4.2.8 (Master reduction strategy)**.** We define a normalization algorithm $\mathcal{N}(t)$ taking as input a typed term t and producing a term t' such that $t \mapsto_{\mathsf{g}}^* t'$. Assume that the free variables of t are $x_1^{A_1}, \ldots, x_n^{A_n}$ and let $\mathcal{A}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \ldots, A_n$. The algorithm performs the following operations.

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

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

3. If t is a redex, we select the *smallest* subterm w of t having maximal $\mathcal{A}$-communication-complexity r. A sequence of terms is produced

$$w \succ_{\mathsf{g}} w_1 \succ_{\mathsf{g}} w_2 \succ_{\mathsf{g}} \ldots \succ_{\mathsf{g}} w_n$$

   such that w_n has $\mathcal{A}$-communication-complexity strictly smaller than r. We substitute w_n for w in t obtaining t' and recursively execute $\mathcal{N}(t')$.

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

Proposition 4.2.5. *Let $t : A$ be any term. Then $t \mapsto_{\mathsf{g}}^* t'$, where t' is a parallel form.*

Proof. Easy structural induction on t. See Proposition 4.1.8. □

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

Lemma 4.2.6. *Let $t : A$ be a term in parallel form which is not a simply typed λ-term and $\mathcal{A}$ containing all proper subformulae of A and closed under subformulae. Assume that $r > 0$ is the maximum $\mathcal{A}$-communication-complexity of the subterms of t. Assume that the free variables $x_1^{A_1}, \ldots, x_n^{A_n}$ of t are such that for every i, either each strong subformula of A_i is in $\mathcal{A}$, or each proper prime subformula of A_i is in $\mathcal{A}$ or has at most r symbols. Suppose moreover that no subterm $u \parallel_a v$ with $\mathcal{A}$-communication-complexity r contains a subterm of the same $\mathcal{A}$-communication-complexity. Then there exists t' such that $t \succ_{\mathtt{g}}^* t'$ and the maximal among the $\mathcal{A}$-communication-complexity of the subterms of t' is strictly smaller than r.*

Proof. We prove the lemma by lexicographic induction on the pair

$$(\rho, k)$$

where k is the number of subterms of t with maximal $\mathcal{A}$-complexity ρ among those with $\mathcal{A}$-communication-complexity r.

Let $u \parallel_a v$ be the *smallest* subterm of t having $\mathcal{A}$-complexity ρ. Four cases can occur.

(a) $\rho = (r, d, l, o)$, with $d > 0$. We first show that the term $u \parallel_a v$ is a redex. Now, the free variables of $u \parallel_a v$ are among $x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{B_1 \to C_1}, \ldots, a_m^{B_m \to C_m}$ and the communication kind of a is C, D.

Hence, suppose by contradiction that all the prime factors of C and D are proper subformulae of A or strong subformulae of one among $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_m \to C_m$. Given that $r > 0$ there is a prime factor P of C or D such that P has r symbols and does not belong to $\mathcal{A}$. The possible cases are two: (i) P is a proper subformula of a prime proper subformula A_i' of A_i such that $A_i' \notin \mathcal{A}$; (ii) P, by Proposition 4.1.2, is a proper subformula of a prime factor of B_i or C_i. If (i), then the number of symbols of A_i' is less than or equal to r, so P cannot be a proper subfomula of A_i', which is a contradiction. If (ii), then, since by hypothesis $a_i^{B_i \to C_i}$ is bound in t, there is a prime factor of B_i or C_i having a number of symbols greater than r, hence we conclude that there is a subterm $w_1 \parallel_{a_i} w_2$ of t having $\mathcal{A}$-complexity greater than ρ, which is absurd.

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

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

is then a redex of t and by replacing it with

$$(w_1 \parallel_a v) \parallel_b (w_2 \parallel_a v) \tag{4.2}$$

we obtain from t a term t' such that $t \succ_{\mathtt{g}} t'$ according to Definition 4.2.7. We must verify that we can apply to t' the main induction hypothesis. Indeed, the reduction $t \succ_{\mathtt{g}} t'$ duplicates all the subterms of v, but all of their $\mathcal{A}$-complexities are smaller than r, because $u \parallel_a v$ by choice is the smallest subterm of t having maximal $\mathcal{A}$-complexity ρ. The two terms $w_1 \parallel_a v$ and $w_2 \parallel_a v$ have smaller $\mathcal{A}$-complexity than ρ, because they have numbers

of occurrences of the symbol $\|$ strictly smaller than in $u \parallel_a v$. Moreover, the terms in t' with (4.2) as a subterm have, by hypothesis, $\mathcal{A}$-communication-complexity smaller than r and hence $\mathcal{A}$-complexity smaller than ρ. Assuming that the communication kind of b is F, G, the prime factors of F and G that are not in $\mathcal{A}$ must have fewer symbols than the prime factors of C and D that are not in $\mathcal{A}$, again because $u \parallel_a v$ by choice is the smallest subterm of t having maximal $\mathcal{A}$-complexity ρ; hence, the $\mathcal{A}$-complexity of (4.2) is smaller than ρ. Therefore the number of subterms of t' with $\mathcal{A}$-complexity ρ is strictly smaller than k. By induction hypothesis, $t' \succ_{\mathsf{g}}^* t''$, where t'' satisfies the thesis.

(b) $\rho = (r, d, l, o)$, with $d = 0$ and $l > 0$. Since $d = 0$, u and v are simply typed λ-terms – and thus strongly normalizable [GLT89] – so we may assume $u \mapsto_{\mathsf{g}}^* u' \in \mathrm{NF}_{\mathsf{g}}$ and $v \mapsto_{\mathsf{g}}^* v' \in \mathrm{NF}_{\mathsf{g}}$ by a sequence intuitionistic reduction rules. By replacing in t the subterm $u \parallel_a v$ with $u' \parallel_a v'$, we obtain a term t' such that $t \succ_{\mathsf{g}} t'$ according to Definition 4.2.7. Moreover, the terms in t' with $u' \parallel_a v'$ as a subterm have, by hypothesis, $\mathcal{A}$-communication-complexity smaller than r and hence $\mathcal{A}$-complexity is smaller than ρ. By induction hypothesis, $t' \succ_{\mathsf{g}}^* t''$, where t'' satisfies the thesis.

(c) $\rho = (r, d, l, o)$, with $d = l = 0$. Since $d = 0$, u and v are simply typed λ-terms. Since $l = 0$, u and v are in normal form and thus satisfy conditions 1. and 2. of Proposition 4.1.4. We need to check that $u \parallel_a v$ is a redex, in particular that the communication complexity of a is greater than 0. Assume that the free variables of $u \parallel_a v$ are among $x_1^{A_1}, \dots, x_n^{A_n}, a_1^{B_1 \to C_1}, \dots, a_m^{B_m \to C_m}$ and that the communication kind of a is C, D. As we argued above, we obtain that not all the prime factors of C and D are proper subformulae of A or strong subformulae of one among $A_1, \dots, A_n, B_1 \to C_1, \dots, B_m \to C_m$. By Definition 4.2.3, $u \parallel_a v$ is a redex.

We now prove that every occurrence of a in u and v is of the form $a\,\xi$ for some term or projection ξ. First of all, a occurs with arrow type both in u and v. Moreover, $u : A$ and $v : A$, since $t : A$ and t is a parallel form; hence, the types $C \to D$ and $D \to C$ cannot be subformulae of A, otherwise $r = 0$, and cannot be proper subformulae of one among $A_1, \dots, A_n, B_1 \to C_1, \dots, B_n \to C_n$, otherwise the prime factors of C, D would be strong subformulae of one among $A_1, \dots, A_n, B_1 \to C_1, \dots, B_m \to C_m$. Thus by Proposition 4.1.5 we are done. Two cases can occur.

- a does not occur in u or v: to fix ideas, let us say it does not occur in u. By performing a simplification reduction, we replace in t the term $u \parallel_a v$ with u and obtain a term t' such that $t \succ_{\mathsf{g}} t'$ according to Definition 4.2.7. After the replacement, the number of subterms having maximal $\mathcal{A}$-complexity ρ in t' is strictly smaller than the number of such subterms in t. By induction hypothesis, $t' \succ_{\mathsf{g}}^* t''$, where t'' satisfies the thesis.

- a occurs in u and in v. Let $u = \mathcal{C}[a\,w_1\,\sigma]$ and $v = \mathcal{D}[a\,w_2\,\tau]$ where the displayed occurrences of a are the rightmost in u and v and σ, τ are the stacks of *all* terms or projections a is applied to. By applying a basic cross reduction to $\mathcal{C}[a\,w_1\,\sigma] \parallel_a \mathcal{D}[a\,w_2\,\tau]$ we obtain either the term $\mathcal{C}[a\,w_1\sigma] \parallel_a \mathcal{D}[w_1\tau]$ or the term

$\mathcal{C}[w_2\sigma] \parallel_a \mathcal{D}[a\, w_2\tau]$ and in both cases we obtain a term with $\mathcal{A}$-complexity $\rho' = (r, d, l, o')$ where $o' < o$.

If we apply a cross reduction, we obtain the term $(*)$

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

where $\boldsymbol{y}$ is the sequence of the free variables of w_1 which are bound in $\mathcal{C}[a\, w_1\, \sigma]$ and $\boldsymbol{z}$ is the sequence of the free variables of w_2 which are bound in $\mathcal{D}[a\, w_2\, \tau]$ and a does not occur neither in w_1 nor in w_2. Since u, v satisfy conditions 1. and 2. of Proposition 4.1.4 the types $Y_1, \ldots, Y_i$ and $Z_1, \ldots, Z_j$ of respectively the variables $\boldsymbol{y}$ and $\boldsymbol{z}$ are proper subformulae of A or strong subformulae of the formulae $A_1, \ldots, A_n, B_1 \to C_1, \ldots, B_m \to C_m$. Hence, the types among $Y_1, \ldots, Y_i, Z_1, \ldots, Z_j$ which are not in $\mathcal{A}$ are strictly smaller than all the prime factors of the formulae $B_1, C_1, \ldots, B_m, C_m$. Since the communication kind of b is $Y_1 \wedge \ldots \wedge Y_i, Z_1 \wedge \ldots \wedge Z_j$, by Definition 4.2.6 either the $\mathcal{A}$-complexity of the term $(*)$ above is strictly smaller than the $\mathcal{A}$-complexity ρ of $u \parallel_a v$, or the communication kind of b is $\top$. In the latter case we apply a simplification reduction $u_1 \parallel_b v_1 \mapsto_{\mathtt{g}} u_1$ or $u_1 \parallel_b v_1 \mapsto_{\mathtt{g}} v_1$ and obtain a term with $\mathcal{A}$-complexity strictly smaller than ρ.

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

$$w_1^{b\langle \boldsymbol{z}\rangle/\boldsymbol{y}}\,\tau \mapsto_{\mathtt{g}}^* w_1' \in \mathrm{NF}_{\mathtt{g}} \text{ and } w_2^{b\langle \boldsymbol{y}\rangle/\boldsymbol{z}}\,\sigma \mapsto_{\mathtt{g}}^* w_2' \in \mathrm{NF}_{\mathtt{g}}$$

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

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

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

$$(\mathcal{D}[w_1'] \parallel_a \mathcal{C}[a\, w_1\sigma]) \parallel_b (\mathcal{C}[w_2'] \parallel_a \mathcal{D}[a\, w_2\tau]) \tag{4.3}$$

By construction $t \succ_{\mathtt{g}} t'$. Moreover, the terms in t' with (4.3) as a subterm have, by hypothesis, $\mathcal{A}$-communication-complexity smaller than r and hence $\mathcal{A}$-complexity is smaller than ρ. Hence, we can apply the main induction hypothesis to t' and obtain by induction hypothesis, $t' \succ_{\mathtt{g}}^* t''$, where t'' satisfies the thesis.

(d) $\rho = (r, d, l, o)$, with $d = l = o = 0$. Since $o = 0$, $u \parallel_a v$ is a redex. To fix ideas, let us say a does not occur in u. By performing a simplification reduction, we replace $u \parallel_a v$ with u so that $u \parallel_a v \succ_{\mathtt{g}} u$ according to Definition 4.2.7. Hence, by induction hypothesis, $t' \succ_{\mathtt{g}}^* t''$, where t'' satisfies the thesis. □

Proposition 4.2.7. *Let $t : A$ be any term in parallel form. Then $t \mapsto_{\mathsf{g}}^* t'$, where t' is a parallel normal form.*

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

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

where $|\mathcal{A}|$ is the cardinality of $\mathcal{A}$, r is the maximal $\mathcal{A}$-communication-complexity of the subterms of t, k is the number of subterms of t having maximal $\mathcal{A}$-communication-complexity r and s is the size of t. If t is a simply typed λ-term, it has a normal form [GLT89] and we are done; so we assume t is not. There are two main cases.

First case: *t is not a redex.* Let $t = u \parallel_a v$ and let B, C be the communication kind of a. Then, the communication complexity of a is 0 and by Definition 4.2.3 every prime factor of B or C belongs to $\mathcal{A}$. Let $\mathcal{A}'$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \dots, A_n, B \to C$; let $\mathcal{A}''$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \dots, A_n, C \to B$. By Proposition 4.1.2, every strong subformula of $B \to C$ or $C \to B$ is a proper subformula of a prime factor of B or C, and this prime factor is in $\mathcal{A}$. Hence, $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{A}'' \subseteq \mathcal{A}$.

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

If $\mathcal{A}' \subset \mathcal{A}$, again by induction hypothesis $u \mapsto_{\mathsf{g}}^* u'$, where u' is a normal parallel form.

The very same argument on $\mathcal{A}''$ shows that $v \mapsto_{\mathsf{g}}^* v'$, where v' is a normal parallel form.

Let now $t' = u' \parallel_a v'$, so that $t \mapsto_{\mathsf{g}}^* t'$. If t' is normal, we are done. If t' is not normal, since u' and v' are normal, the only possible redex remaining in t' is the whole term itself, i.e., $u' \parallel_a v'$: that happens only if the free variables of t' are fewer than those of t; w.l.o.g., assume they are $x_1^{A_1}, \dots, x_i^{A_i}$, with $i < n$. Let $\mathcal{B}$ be the set of the proper subformulae of A and the strong subformulae of the formulae $A_1, \dots, A_i$. Since t' is a redex, the communication complexity of a is greater than 0; by Definition 4.2.3, a prime factor of B or C is not in $\mathcal{B}$, so we have $\mathcal{B} \subset \mathcal{A}$. By induction hypothesis, $t' \mapsto_{\mathsf{g}}^* t''$, where t'' is a parallel normal form.

Second case: *t is a redex.* Let $u \parallel_a v$ be the *smallest* subterm of t having $\mathcal{A}$-communication-complexity r. The free variables of $u \parallel_a v$ satisfy the hypotheses of Lemma 4.2.6 either because have type A_i and $\mathcal{A}$ contains all the strong subformulae of A_i, or because the prime proper subformulae of their type have at most r symbols, by maximality of r. By Lemma 4.2.6 $u \parallel_a v \succ_{\mathsf{g}}^* w$ where the maximal among the $\mathcal{A}$-communication-complexity of the subterms of w is strictly smaller than r. Let t' be the term obtained replacing w

for $u \parallel_a v$ in t. We can now apply the induction hypothesis and obtain $t' \mapsto_{\mathsf{g}}^* t''$ with t'' in parallel normal form. □

The normalization for λ_{G}, and thus for N$\mathfrak{G}$, easily follows.

Theorem 4.2.8. *Suppose that $t : A$ is a proof term of* **GL**. *Then $t \mapsto_{\mathsf{g}}^* t' : A$, where t' is a normal parallel form.*

4.2.6 The expressive power of λ_{G}

We show now that λ_{G} is more expressive than simply typed λ-calculus and then we present some examples of λ_{G} programs.

Example 4.2.2 (**Parallel OR**)**.** Berry's sequentiality theorem (see [GLT89]) implies that there is no λ-term $\mathsf{O} : \mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool}$ such that $\mathsf{O}\mathtt{ff}\mathtt{ff} \mapsto_{\mathsf{g}} \mathtt{ff}$, $\mathsf{O}u\mathtt{tt} \mapsto_{\mathsf{g}} \mathtt{tt}$, $\mathsf{O}\mathtt{tt}u \mapsto_{\mathsf{g}} \mathtt{tt}$, where u is an arbitrary normal term, and thus possibly a variable.

The λ_{G}-term for such parallel OR is (as usual the term "$\mathsf{if}\, u \,\mathsf{then}\, s \,\mathsf{else}\, t$" reduces to s if $u = \mathtt{tt}$, and to t if $u = \mathtt{ff}$):

$$\mathsf{O} := \lambda x^{\mathtt{Bool}}\, \lambda y^{\mathtt{Bool}}\, (\mathsf{if}\, x \,\mathsf{then}\, (\lambda z\, \lambda k\, z) \,\mathsf{else}\, (\lambda z\, \lambda k\, k))\mathtt{tt}(ax)$$
$$\parallel_a (\mathsf{if}\, y \,\mathsf{then}\, (\lambda z\, \lambda k\, z) \,\mathsf{else}\, (\lambda z\, \lambda k\, k))\mathtt{tt}(ay)$$

where the communication kind of a is $\mathtt{Bool}, \mathtt{Bool}$. Now

$$\begin{aligned}\mathsf{O}\, u\, \mathtt{tt} &\mapsto_{\mathsf{g}}^* (\mathsf{if}\, u \,\mathsf{then}\, (\lambda z\, \lambda k\, z) \,\mathsf{else}\, (\lambda z\, \lambda k\, k))\mathtt{tt}(au) \\ &\quad \parallel_a (\mathsf{if}\, \mathtt{tt} \,\mathsf{then}\, (\lambda z\, \lambda k\, z) \,\mathsf{else}\, (\lambda z\, \lambda k\, k))\mathtt{tt}(a\mathtt{tt}) \\ &\mapsto_{\mathsf{g}}^* (\mathsf{if}\, u \,\mathsf{then}\, (\lambda z\, \lambda k\, z) \,\mathsf{else}\, (\lambda z\, \lambda k\, k))\mathtt{tt}(au) \parallel_a \mathtt{tt} \mapsto_{\mathsf{g}} \mathtt{tt}\end{aligned}$$

And symmetrically $\mathsf{O}\,\mathtt{tt}\, u \mapsto_{\mathsf{g}}^* \mathtt{tt}$. On the other hand

$$\begin{aligned}\mathsf{O}\,\mathtt{ff}\,\mathtt{ff} \mapsto_{\mathsf{g}}^* &\quad (\lambda z\, \lambda k\, k)\mathtt{tt}(a\mathtt{ff}) \parallel_a (\lambda z\, \lambda k\, k)\mathtt{tt}(a\mathtt{ff}) \\ \mapsto_{\mathsf{g}}^* &\quad (a\mathtt{ff}) \parallel_a (a\mathtt{ff}) \\ \mapsto_{\mathsf{g}}^* &\quad (\mathtt{ff} \parallel_a (a\mathtt{ff})) \parallel_b (\mathtt{ff} \parallel_a (a\mathtt{ff})) \quad \mapsto_{\mathsf{g}}^* \quad \mathtt{ff}\end{aligned}$$

In the following example of computation in λ_{G}, similar to one shown in [CP10a], we simulate the communication needed to conclude an online sale.

Example 4.2.3 (**Buyer and vendor**)**.** We model the following transaction: a buyer tells a vendor a product name $\mathsf{prod} : \mathtt{String}$, the vendor computes the value $\mathsf{price} : \mathbb{N}$ of prod and sends it to the buyer, the buyer sends back the credit card number $\mathsf{card} : \mathtt{String}$ which is used to pay.

We introduce the following functions: $\mathsf{cost} : \mathtt{String} \to \mathbb{N}$ with input a product name prod and output its cost price; $\mathsf{pay\_for} : \mathbb{N} \to \mathtt{String}$ with input a price and output a

credit card number card; $\mathsf{use} : \mathtt{String} \to \mathbb{N}$ that obtains money using as input a credit card number $\mathsf{card} : \mathtt{String}$. The buyer and the vendor are the contexts $\mathcal{B}$ and $\mathcal{V}$ of type $\mathtt{Bool}$. Notice that the terms representing buyer and vendor exchange their position at each cross reduction. For a of kind $\mathtt{String}, \mathbb{N}$, the program is:

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

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

4.2.7 Disjunction in Gödel–Dummett logic

We formally define disjunction and disjunction rules in N$\mathfrak{G}$ and the corresponding computational constructs of λ_{G}. We employ here the same method used in Section 4.1.5 for N$\mathfrak{Cl}$ and λ_{Cl}, the main difference being the following definition of $\vee$.

Definition 4.2.9 (Definition of $\vee$ in **GL**)**.** For any formula A, we define the translation $A^{\vee g}$ inductively as follows:

- for any propositional variable p, $p^{\vee g} = p$
- $\bot^{\vee g} = \bot$
- $(B \to C)^{\vee g} = B^{\vee g} \to C^{\vee g}$
- $(B \wedge C)^{\vee g} = B^{\vee g} \wedge C^{\vee g}$
- $(B \vee C)^{\vee g} = ((B^{\vee g} \to A^{\vee g}) \to A^{\vee g}) \wedge ((A^{\vee g} \to B^{\vee g}) \to B^{\vee g})$

For any set of formulae Γ, we denote by $\Gamma^{\vee g}$ the set $\{A^{\vee g} : A \in \Gamma\}$.

We show the completeness of N$\mathfrak{G}$ with respect to the translation of **GL** theorems by ${}^{\vee g}$. The proof proceeds as that of Proposition 4.1.12 for N$\mathfrak{Cl}$, but the cases concerning the disjunction rules differ due to the differences between ${}^{\vee g}$ and the translation ${}^{\vee c}$ of Definition 4.1.16.

Proposition 4.2.9. *For any set Γ of formulae and formula A, if $\Gamma \vdash_{\mathbf{GL}} A$, then $\Gamma^{\vee g} \vdash_{\mathrm{N}\mathfrak{G}} A^{\vee g}$.*

Proof. By Theorem 4.2.1, there is a derivation $\mathcal{D}$ of A from Γ in N$\mathfrak{J}$ + (Lin). The proof is by induction on the height l of $\mathcal{D}$.

If $\mathcal{D}$ has height 0, then it is A and our N$\mathfrak{G}$ proof is just $A^{\vee g}$.

We reason now by cases on the last rule applied in $\mathcal{D}$:

- $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ C\end{array}}{B \to C}$$

  with assumptions Γ. By induction hypothesis there exists a derivation

  $$\begin{array}{c}\mathcal{E}\\ C^{\vee g}\end{array}$$

  in N$\mathfrak{G}$ with assumptions $\Gamma^{\vee g}$ and possibly $B^{\vee g}$. We construct the derivation

  $$\dfrac{\begin{array}{c}\mathcal{E}\\ C^{\vee g}\end{array}}{B^{\vee g} \to C^{\vee g}}$$

  possibly discharging $B^{\vee g}$ and obtain the required derivation of $A^{\vee g}$ in N$\mathfrak{G}$ since $(B \to C)^{\vee g} = B^{\vee g} \to C^{\vee g}$.

- $$\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ B \to C & B\end{array}}{C}$$

  By induction hypothesis there exist derivations

  $$\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2\\ (B \to C)^{\vee g} & B^{\vee g}\end{array}$$

  in N$\mathfrak{G}$ whose assumptions are contained in $\Gamma^{\vee g}$. Since $(B \to C)^{\vee g} = B^{\vee g} \to C^{\vee g}$ the required derivation is

  $$\dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2\\ B^{\vee g} \to C^{\vee g} & B^{\vee g}\end{array}}{C^{\vee g}}$$

- $$\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ B & C\end{array}}{B \wedge C}$$

  By induction hypothesis there exist derivations

  $$\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2\\ B^{\vee g} & C^{\vee g}\end{array}$$

  in N$\mathfrak{G}$ whose assumptions are contained in $\Gamma^{\vee g}$. Since $(B \wedge C)^{\vee g} = B^{\vee g} \wedge C^{\vee g}$ the required derivation is

  $$\dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2\\ B^{\vee g} & C^{\vee g}\end{array}}{B^{\vee g} \wedge C^{\vee g}}$$

- $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}\\ B_1 \wedge B_2\end{array}}{B_i}$$

By induction hypothesis there exists a derivation

$$\begin{array}{c}\mathcal{E}\\(B_1 \wedge B_2)^{\vee g}\end{array}$$

in N$\mathfrak{G}$ whose assumptions are $\Gamma^{\vee g}$. Since $(B_1 \wedge B_2)^{\vee g} = B_1^{\vee g} \wedge B_2^{\vee g}$ the required derivation is

$$\dfrac{\begin{array}{c}\mathcal{E}\\B_1^{\vee g} \wedge B_2^{\vee g}\end{array}}{B_i^{\vee g}}$$

- $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}\\\bot\end{array}}{P}$$

for some atomic formula $P \neq \bot$. By induction hypothesis there exists a derivation

$$\begin{array}{c}\mathcal{E}\\\bot^{\vee g}\end{array}$$

in N$\mathfrak{G}$ whose assumptions are $\Gamma^{\vee g}$. Since $\bot^{\vee g} = \bot$ and $P^{\vee g} = P$ the required derivation is

$$\dfrac{\begin{array}{c}\mathcal{E}\\\bot^{\vee g}\end{array}}{P^{\vee g}}$$

- $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}\\B_i\end{array}}{B_1 \vee B_2}$$

By induction hypothesis there exists a derivation

$$\begin{array}{c}\mathcal{E}\\B_i^{\vee g}\end{array}$$

in N$\mathfrak{G}$ whose assumptions are $\Gamma^{\vee g}$. Since

$$(B_1 \vee B_2)^{\vee g} = ((B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}) \wedge ((B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g})$$

the required derivation is for $i = 1$:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{E}\\B_1^{\vee g}\end{array}}{(B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}} \quad \dfrac{\dfrac{[B_1^{\vee g} \to B_2^{\vee g}]^1 \quad \begin{array}{c}\mathcal{E}\\B_1^{\vee g}\end{array}}{B_2^{\vee g}}}{(B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g}}\,{}^1}{((B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}) \wedge ((B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g})}$$

and for $i = 2$:

$$\dfrac{\dfrac{\dfrac{[B_2^{\vee g} \to B_1^{\vee g}]^1 \quad \begin{array}{c}\mathcal{E}\\B_2^{\vee g}\end{array}}{B_1^{\vee g}}}{(B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}}\,{}^1 \quad \dfrac{\begin{array}{c}\mathcal{E}\\B_2^{\vee g}\end{array}}{(B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g}}}{((B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}) \wedge ((B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g})}$$

- $$\mathcal{D} = \dfrac{\begin{array}{ccc}\mathcal{D}_0 & \mathcal{D}_1 & \mathcal{D}_2\\ B_1 \vee B_2 & C & C\end{array}}{C}$$

  By induction hypothesis there exist derivations

  $$\begin{array}{ccc}\mathcal{E}_0 & \mathcal{E}_1 & \mathcal{E}_2\\ (B_1 \vee B_2)^{\vee g} & C^{\vee g} & C^{\vee g}\end{array}$$

  in N$\mathfrak{G}$ where the assumptions of $\mathcal{E}_0$ are included in $\Gamma^{\vee g}$, and the assumptions of each $\mathcal{E}_i$ for $i \in \{1,2\}$ are included in $\Gamma^{\vee g}$ plus possibly $B_i^{\vee g}$. Since

  $$(B_1 \vee B_2)^{\vee g} = ((B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}) \wedge ((B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g})$$

  the required derivation is

  $$\dfrac{\dfrac{\dfrac{\mathcal{E}_0}{((B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}) \wedge ((B_1^{\vee g} \to B_2^{\vee g}) \to B_2^{\vee g})}}{(B_2^{\vee g} \to B_1^{\vee g}) \to B_1^{\vee g}} \qquad \dfrac{[B_2^{\vee g} \to B_1^{\vee g}]^1}{\begin{array}{c}\vdots\\ B_2^{\vee g}\end{array}}}{\begin{array}{cc}B_1^{\vee g} & \\ \mathcal{E}_1 & \\ C^{\vee g} & \end{array} \qquad \begin{array}{c}\\ \mathcal{E}_2\\ C^{\vee g}\end{array}}{C^{\vee g}}\ (\mathrm{Lin})^1$$

  where by $\begin{array}{c}\vdots\\ B_1^{\vee g}\\ \mathcal{E}_1\end{array}$ we denote the derivation obtained from $\mathcal{E}_1$ deriving as shown all occurrences of the assumption $B_1^{\vee g}$. The derivations of the occurrences of $B_2^{\vee g}$ in $\mathcal{E}_2$ are symmetrically constructed possibly using $[B_1^{\vee g} \to B_2^{\vee g}]^1$.

  □

We define now the computational constructs of case distinction and injection in λ_{G}.

We can define the λ_{G}-terms $\iota_0(u)$, $\iota_1(u)$ and $t[x_0.v_0, x_1.v_1]$ such that for $i \in \{0,1\}$ we have

$$\iota_i(u)[x_0.v_0, x_1.v_1] \ \mapsto_{\mathsf{g}}\ v_i[u/x_i]$$

as follows. Let $A \vee B := ((B \to A) \to A) \wedge ((A \to B) \to B)$

$$\iota_0(u) \ :=\ \langle \lambda x^{B\to A}\, u, \lambda y^{A\to B}\, yu\rangle : A \vee B \qquad \iota_1(u) \ :=\ \langle \lambda y^{B\to A}\, yu, \lambda x^{A\to B}\, u\rangle : A \vee B$$

$$t\,[x_0.v_0, x_1.v_1] \ :=\ v_0[t\,\pi_0\,a^{B\to A}/x_0] \parallel_a v_1[t\,\pi_1\,a^{A\to B}/x_1] : F$$

where x and y do not occur in u, a is a fresh channel, $v_0 : F$, $v_1 : F$, and $t : A \vee B$. For instance we can verify that

$$\begin{gathered}\iota_0(u)\,[x_0.v_0, x_1.v_1] \ :=\\ v_0[\langle \lambda x^{B\to A}\, u, \lambda y^{A\to B}\, yu\rangle \pi_0\, a/x_0] \parallel_a v_1[\langle \lambda x^{B\to A}\, u, \lambda y^{A\to B}\, yu\rangle \pi_1\, a/x_1]\\ \mapsto^*_{\mathsf{g}} \quad v_0[(\lambda x^{B\to A}\, u)\, a/x_0] \parallel_a v_1[(\lambda y^{A\to B}\, yu)\, a/x_1]\\ \mapsto_{\mathsf{g}} \quad v_0[u/x_0] \parallel_a v_1[a\,u/x_1] \quad \mapsto_{\mathsf{g}} \quad v_0[u/x_0]\end{gathered}$$

4.3 Comparison between λ_{Cl}, λ_{G} and related calculi

The most widespread formalisms to represent concurrent computation is π-calculus. The main difference between π-calculus and the calculi λ_{Cl} and λ_{G} is in the purpose of the systems: while π-calculus [Mil92, SW03] is a formalism for *modeling* concurrent systems, the calculi λ_{Cl} and λ_{G} are concurrent *functional languages*. Nonetheless, there are relevant connections as well.

The first similarity between these calculi and π-calculus lies in the channel restrictions: the a occurring in $u \parallel_a v$ and the a in a π-calculus term $\nu a(P \mid Q)$ have the same rôle: designating a private communication channel between two parallel processes. In λ_{Cl} moreover, the result of communicating a closed process or data is similar to the result of asynchronous π-calculus [HT91] communications $\overline{a}\,(x) \mid a(y).Q \mapsto Q[x/y]$.

The main difference between the communication mechanism of π-calculus and that of our concurrent λ-calculi is that the former only supports the transmission of data or channel names, while the latter supports higher-order communication. The communication mechanism of λ_{Cl} and λ_{G}, moreover, is very general and can handle not only closed and open processes, but also processes that are closed in their original environment, but become open after the communication. Such λ-calculi, in other terms, have mechanisms for handling the transmission of function closures, which is a well known problem in the study of code mobility, see for instance [EBPJ11]. While extensions of π-calculus with higher-order communication have been defined [San93], no natural logic-based type systems for such an extension exist.

The number of recipients of a communication is another difference between π-calculus and λ_{Cl}. While in π-calculus only one process can receive each message, in λ_{Cl} we can have broadcast communications: $\mathcal{C}[\overline{a}\ u] \parallel_a (\mathcal{D}_1 \parallel \ldots \parallel \mathcal{D}_m) \mapsto_{\mathsf{c}} \mathcal{D}_1[u/a] \parallel \ldots \parallel \mathcal{D}_m[u/a]$.

As for restrictions on the connections of processes by communication channels, π-calculus has none but λ_{Cl} and λ_{G} have rather strict symmetry conditions. Similar conditions are adopted in typed versions of π-calculus, see [TCP13, Wad12].

Finally, in pure π-calculus both sender and recipient of a communication might be selected non-deterministically. Since the communication of λ_{Cl} is a broadcasting to all recipients, only the sender can be non-deterministically selected: in applying the reduction $(\mathcal{C}_1[\overline{a}\ t_1] \parallel \ldots \parallel \mathcal{C}_n[\overline{a}\ t_n]) \parallel_a \mathcal{D} \mapsto_{\mathsf{c}} \mathcal{D}[t_i/a]$, for instance, we can choose any $\mathcal{C}_i[\overline{a}\ t_i]$ for $i \in \{1, \ldots, n\}$. In λ_{G} we have no non-deterministic choice in this respect.

Several untyped [Bou89] and typed λ-calculi [Wad12, TCP13] have been introduced extending λ-calculus by primitives for π-calculus-style communications [CP10b]. Since the expressive power of these calculi is closer to that of $\lambda_{\parallel}$, introduced in Chapter 5, we refer the reader to Section 5.7.

As for a comparison of the two calculi presented in this chapter, the main difference between them is that λ_{Cl} cannot encode a *dialogue* between two processes – if a process u receives a message from a process v, then v cannot send a message to u – but the

bidirectional channels of λ_{G} enable such exchanges. The communication reductions of λ_{Cl}, on the other hand, are simpler than those of λ_{G} precisely because the communication variables that can receive messages cannot send messages. A consequence of this is, for example, that we can implement broadcasting communications in λ_{Cl} while we cannot implement them in λ_{G}.

Finally, the type system of λ_{Cl} is the same as that of λ_{exn}, which we discuss in Section 2.4.2. Nevertheless, firstly, the computational interpretation provided by λ_{exn} is not in terms of concurrent computation but in terms of exception handling and, secondly, the normalization of λ_{exn} does not imply that normal terms enjoy the subformula property. In particular, it seems that λ_{exn} is missing some reduction rules corresponding to the communications of terms that depend on their computational environment.

CHAPTER 5

A typed parallel λ-calculus based on disjunctive tautologies

As we have seen in Chapter 4, λ_{Cl} and λ_{G} only have channels connecting pairs of processes:

It is certainly possible to construct complex networks using several channels of this kind. However, types can only regulate the behavior of *single* channels in the presented systems. As a consequence, the behavior of complex networks, made of *groups* of channels, cannot be logically described. It is clear then that having complex channels – namely, individual channels that connect several processes and ultimately represent a whole network – is of crucial importance both for programming and for modeling concurrent systems. As we will see, different axioms corresponding to hypersequent rules can type different complex network topologies as, for example, those shown in Figure 5.1. Hence, by using a class of axioms to define our type systems we can obtain a more fine grained control over the interactions among the processes in a term.

In this chapter we introduce the calculus $\lambda_{\parallel}$ which extends in this direction the computational capabilities of λ_{Cl} and λ_{G} and presents excellent computational properties. On the other hand, in Chapter 6 we will define the family of calculi $\lambda_{\parallel \mathrm{L}}$, which generalizes λ_{Cl} and λ_{G} from a theoretical perspective and confirms Avron's thesis.

The calculus $\lambda_{\parallel}$ is a typed parallel λ-calculus. Its type system is based on a rather general class Ax of disjunctive classical tautologies. The calculus $\lambda_{\parallel}$ is intended as a language to encode parallel algorithms. The channels of $\lambda_{\parallel}$ can model arbitrary network topologies. Furthermore, we present an algorithm to extract typing rules from graph-specified communication topologies in such a way that the typed terms can only communicate

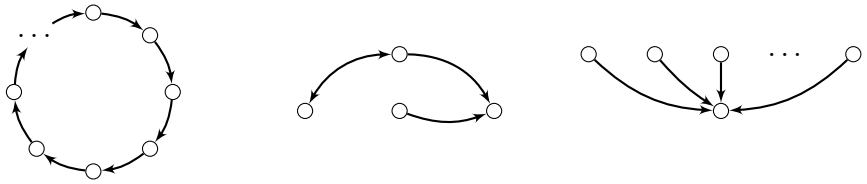


Figure 5.1: $\lambda_{\|}$ multi-party channels.

according to the topology. The usefulness of the direct implementation of network topologies in the context of parallel programming is also witnessed by the development of systems designed with this aim. The tool Grace [HL13] (Graph-based Communication in Eden) for the language Eden [LOP05], for instance, enables programmers to declare and encode network topologies as graphs and provides special constructs to generate the actual process networks: the processes are created and the communication channels are installed.

From a technical point of view, $\lambda_{\|}$ presents two strong simplifications with respect to λ_{Cl} and λ_{G}. First, it only features mini cross reductions – corresponding to basic cross reductions – and hence it only allows the communication of terms that do not depend on their environment. Second, $\lambda_{\|}$-terms contain at most one parallel operator that binds communication variables. These simplifications strongly reduce the non-determinism of the calculus and hence have the crucial effect of enabling us to prove that the evaluation of $\lambda_{\|}$-terms terminates regardless of the employed strategy. Nevertheless, they do not constitute a loss in terms of expressive power as long as parallel algorithms are concerned. Indeed, in order to encode such algorithms we need to be able to represent the parallel structure of the problem, to control the interactions between the processes, and thus to predict the outcome of the computation. In $\lambda_{\|}$ these requirements are met because we can encode as a term any communication topology that can be represented as a directed graph, and we can type it using one application of a rule $(\mathcal{A})$ for some $\mathcal{A} \in \mathrm{Ax}$. Typing the whole network by a unique rule, in turn, give us a very fine-grained control over the order of the communications among its nodes.

In order to formally define the type system for $\lambda_{\|}$, we build on ideas presented in [Avr91, DK00] and we consider the class of classical disjunctive tautologies of the form $\bigvee_{i=0}^{m}(A_i \to B_i)$ where each A_i and B_i is a conjunction of propositional atoms. On the one hand, all formulae in such class can be transformed into hypersequent rules [Avr91] – see Table 2.3 for some examples – on the other hand, the disjunctive tautologies that are interpreted as synchronization schemata in [DK00] belong to this class. As already mentioned and as we will show in Section 5.4, there is a restriction on these formulae that defines a class Ax of very simple axiom schemata that suffices to encode all network topologies that can be represented as reflexive directed graphs. An axiom encoding such a graph will correspond in turn to a typing rule that allows the terms to communicate only by channels represented in the graph. We define the type system of $\lambda_{\|}$ as containing a type assignment rule for each axiom in Ax. We can then extract type assignment rules from

graphs in such a way that the communications inside a term typed by such a rule must comply with the network represented by the corresponding graph.

Consider, for instance, the type assignment rules corresponding to the axiom schemata $(A \to A \wedge \bot) \vee (B \to B \wedge A)$ and $(A \to A \wedge B) \vee (B \to B \wedge C) \vee (C \to C \wedge A)$ which are intuitionistically equivalent to the axiom schemata EM and $\mathbf{C}_3$ – see Table 2.3 – respectively:

$$\dfrac{\begin{array}{cc}[a : A \to A \wedge \bot] & [a : B \to B \wedge A] \\ \vdots & \vdots \\ u : C & v : C\end{array}}{{}_a(u \parallel v) : C}\ (\mathrm{EM}') \qquad \dfrac{\begin{array}{ccc}[a : A \to A \wedge B] & [a : B \to B \wedge C] & [a : C \to C \wedge A] \\ \vdots & \vdots & \vdots \\ t : D & u : D & v : D\end{array}}{{}_a(t \parallel u \parallel v) : D}\ (\mathbf{C}_3')$$

These rules establish the communication channels connecting the processes u, v and t, u, v respectively. Notice that since parallelism operators that bind communication variables are no more binary, we adopt the notation ${}_a(\ldots \parallel \ldots \parallel \ldots)$ instead of $\ldots \|_a \ldots$. The (EM') rule above, for example, types a communication topology in which the process v can receive a message of type A from the process u:

$${}_a(\mathcal{C}[a\, s] \parallel \mathcal{D}[a\, t]) \ \mapsto_{\mathtt{p}}\ {}_a(\mathcal{C}[a\, s] \parallel \mathcal{D}[\langle t, s\rangle])$$

The rule $(\mathbf{C}_3')$ above, on the other hand, corresponds to a topology in which t can receive a message from u, u from v and v from t. This reductions bear a resemblance to the cross reductions of λ_{Cl} or λ_{G}, but there are some relevant differences. First of all, as already mentioned, we only transmit terms that do not depend on their computational environment and thus we never introduce a new communication channel to restore violated dependencies. Second, the received message neither replaces all occurrences of the receiving channel a, as in λ_{Cl}, nor erases the argument of the receiving channel, as in λ_{G}. And finally, even if more than one process contains a channel occurrence that could act as receiver, only one process receives messages during a single communication reduction. As λ_{Cl} and λ_{G}, on the other hand, also $\lambda_{\|}$ is strictly more expressive than simply typed λ-calculus.

The present chapter is based on [ACG19b], and structured as follows. We present a type system and reduction rules for $\lambda_{\|}$ in Sections 5.1 and 5.2 respectively. In Section 5.4 we describe a method for automatically extracting $\lambda_{\|}$ typing rules from any graph-based communication, and in Section 5.5 we show the strong normalization proof. In Section 5.6 we showcase the computational capabilities of $\lambda_{\|}$ by presenting $\lambda_{\|}$-terms implementing parallel OR, a parallel algorithm for computing π, and the parallel version of the Floyd–Warshall algorithm to compute the shortest path between two nodes in a graph. In Section 5.7 we compare $\lambda_{\|}$ with similar systems.

5.1 The type system of $\lambda_{\|}$

We consider the class Ax of axiom schemata

$$(A_1 \to A_1 \wedge B_1) \vee \ldots \vee (A_m \to A_m \wedge B_m)$$

$$\frac{t_1 : A \quad \dots \quad t_n : A}{t_1 \parallel \dots \parallel t_n : A}\ (contr)$$

where $t_1, \dots, t_n$ are simply typed λ-terms

$$\frac{\begin{array}{ccc}[a^{A_1 \to A_1 \wedge B_1} : A_1 \to A_1 \wedge B_1] & & [a^{A_m \to A_m \wedge B_m} : A_m \to A_m \wedge B_m] \\ \vdots & & \vdots \\ u_1 \parallel \dots \parallel u_n : B & \dots & u_p \parallel \dots \parallel u_q : B\end{array}}{{}^{\mathcal{A}}_{a}((u_1 \parallel \dots \parallel u_n) \parallel \dots \parallel (u_p \parallel \dots \parallel u_q)) : B}\ (\mathcal{A})$$

where $(A_1 \to A_1 \wedge B_1) \vee \dots \vee (A_m \to A_m \wedge B_m)$ is an instance of $\mathcal{A} \in \mathrm{Ax}$

Table 5.1: Type assignments for $\lambda_{\parallel}$

where $A_1, \dots, A_m$ are pairwise different propositional variables and $B_1, \dots, B_m$ are either $\bot$ or conjunctions of distinct propositional variables among $A_1, \dots, A_m$. Formally, $A_i \neq A_j$ for $i \neq j$; and for any $i \in \{1, \dots, m\}$, either $B_i = \bot$ or $B_i = A_{k_1} \wedge \dots \wedge A_{k_p}$ with $A_{k_i} \neq A_{k_j}$ for $i \neq j$.

Remark 5.1. Each disjunct $A_i \to A_i \wedge B_i$ is logically equivalent to $A_i \to B_i$. Nevertheless, we keep the second, logically redundant, occurrence of A_i in order to type a memory mechanism for input channels, as we will see in Section 5.2.

Example 5.1.1. The axioms $\mathrm{EM} = A \vee \neg A$, $\mathbf{C}_k = (A_1 \to A_2) \vee \dots \vee (A_n \to A_1)$, $\mathrm{Lin} = (A \to B) \vee (B \to A)$, and $\mathbf{G}_k = (A_1 \to A_2) \vee \dots \vee (A_{n-1} \to A_n) \vee \neg A_n$ for $n \geq 2$, are intuitionistically equivalent to axioms in Ax.

Definition 5.1.1 (Terms of $\lambda_{\parallel}$)**.** The terms of $\lambda_{\parallel}$ are defined by the rules for simply typed λ-calculus in Table 2.5 and by the rules in Table 5.1.

As in the previous chapter, the variables are denoted as x^A, y^A, $z^A, \dots$, a^A, b^A, c^A omitting the type when irrelevant. The variables introduced by the $(\mathcal{A})$ rule will be often denoted with letters $a, b, c, \dots$ but do not constitute a separate class. A variable a that occurs in a term ${}^{\mathcal{A}}_{a}(u_1 \parallel \dots \parallel u_m)$ is called **channel** or **communication variable** and represents a *private* communication channel between the parallel processes. The term ${}^{\mathcal{A}}_{a}(u_1 \parallel \dots \parallel u_m)$ will be denoted by ${}_{a}(u_1 \parallel \dots \parallel u_m)$ when $\mathcal{A}$ is clear from the context or irrelevant. All the free occurrences of a in $u_1, \dots, u_m$ are bound in ${}_{a}(u_1 \parallel \dots \parallel u_m)$. All conventions stated for the simply typed λ-calculus apply to $\lambda_{\parallel}$ too.

The rule $(contr)$ is essential for representing parallel terms that cannot communicate between themselves but can still communicate with other processes. From a logical perspective the rule is redundant, but sound.

From a computational perspective the rules $(\mathcal{A})$ produce terms of the shape ${}^{\mathcal{A}}_{a}(v_1 \parallel \dots \parallel v_m)$ that put in parallel terms $v_1, \dots, v_m$, which are called the **processes** of ${}^{\mathcal{A}}_{a}(v_1 \parallel \dots \parallel v_m)$; each term v_i in turn has the shape $u_1 \parallel \dots \parallel u_k$, where $u_1, \dots, u_k$ are simply typed λ-terms and called the **threads** of v_i. Processes can communicate

with each other through the channel a, whereas threads represent parallel independent subprograms than cannot interact with each other. Informally, in order to establish a communication channel connecting two terms v_i and v_j, we require that $a^{A_i \to A_i \wedge B_i}$ occurs in v_i, $a^{A_j \to A_j \wedge B_j}$ occurs in v_j and A_i is one of the conjuncts forming B_j. On one hand, the argument w of a channel application $a^{A_i \to A_i \wedge B_i}\, w$ will be interpreted as a message of type A_i that must be *transmitted*; on the other hand, the channel application $a^{A_j \to A_j \wedge B_j}\, t$ will *receive* several messages of type B_j and containing w that will replace the whole channel application $a^{A_j \to A_j \wedge B_j}\, t$ upon reception. Thus, in general each channel application may send and receive messages. In order to formalize exactly the relation between a process v_i and all the processes v_j such that v_i can send messages to v_j, it is necessary to look at the structure of the axiom schema $\mathcal{A}$, because its instances may lose information about its general shape. For this purpose, we introduce the concept of outlink.

Definition 5.1.2 (Outlinks)**.** Let ${}^{\mathcal{A}}_{a}(v_1 \parallel \ldots \parallel v_m)$ be a term, where

$$\mathcal{A} = (A_1 \to A_1 \wedge B_1) \vee \ldots \vee (A_m \to A_m \wedge B_m)$$

For any $i, j \in \{1, \ldots, m\}$ and $i \neq j$, we say that the term v_i is **outlinked** to the term v_j if $B_j = A_{k_1} \wedge \ldots \wedge A_i \wedge \ldots \wedge A_{k_p}$.

The direction of the communication and the direction of $\to$ are reversed, because the type of the messages, which are arguments of channel occurrences $a^{A_i \to A_i \wedge B_i}$, must be contained in the type of the applications of the receiving channel $a^{A_j \to A_j \wedge (A_{k_1} \wedge \ldots \wedge A_i \wedge \ldots \wedge A_{k_p})}$.

Each rule $(\mathcal{A})$, stripped from the $\lambda_{\|}$-terms, is equivalent to an instance of the axiom $\mathcal{A}$, which explain the rule's name.

Proposition 5.1.1. *The rule $(\mathcal{A})$ is equivalent to the axiom $\mathcal{A} := (A_1 \to A_1 \wedge B_1) \vee \ldots \vee (A_m \to A_m \wedge B_m)$, that is for any formula P and set of formulae Γ, $\Gamma \vdash P$ in $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\perp} + (\mathcal{A})$ if and only if P is provable from Γ in* **IL** *extended by $\mathcal{A}$.*

Proof. ($\Leftarrow$) In $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\perp}$ with disjunction rules, we can easily derive $\mathcal{A}$ from each assumption $A_i \to A_i \wedge B_i$. We can then use the rule $(\mathcal{A})$ to discharge all hypotheses $A_i \to A_i \wedge B_i$ and prove $\mathcal{A}$. ($\Rightarrow$) $(\mathcal{A})$ can be easily simulated applying one disjunction elimination rule for each $\vee$ in $\mathcal{A}$. □

5.2 Communications in $\lambda_{\|}$

We present the reduction rules of $\lambda_{\|}$ that implement the actual communications. communications can transmit as messages arbitrary simply typed λ-terms, provided their free variables are not bound in the surrounding context. As shown in Section 5.6, this communication mechanism is enough to code interesting parallel programs.

Intuitionistic reductions The usual computational rules for the simply typed λ-calculus represent the operations of applying a function to an argument and extracting a component of a pair [GLT89]. From the logical point of view, they are the standard Prawitz reductions [Pra71] for the natural deduction calculus $\mathrm{N}\mathfrak{I}^{\rightarrow}_{\wedge\perp}$ for **IL**.

Mini cross reductions Their goal is to implement communication, namely to transmit programs, in the form of simply typed λ-terms. Since $\lambda_{\|}$-terms may contain more than one occurrence of a channel application, the first choice to make is which occurrence should contain the next output message. For example, here we have two communicating processes, each one consists of two threads:

$$(*) \qquad {}_a(\ (a\,r \parallel a(x(a\,s)))\ \parallel\ (a\,u \parallel a(y(aw)))\ )$$

where r, s, u, w are simply typed λ-terms not containing a. The first process $a\,r \parallel a(x(a\,s)$ contains three occurrences of the channel a. Let us focus on its second thread $a(x(a\,s))$. The channel application $a(x(a\,s))$ cannot transmit the message $x(a\,s)$, because the channel a might be used with a different type in the second process, so type preservation after reduction would fail. Therefore the only possibility here is to choose the second channel application $a\,s$ as the one containing the output message, in this case s. In general, to make sure that the message does not contain the channel a, it is enough to choose as application occurrence that contains the output message the *rightmost occurrence* of the channel a in the whole process. Hence, in any process, the rightmost thread that contains the channel contains the message.

The second choice is which occurrence of a channel application should receive the current message. For example, in the term $(*)$ above the second process $a\,u \parallel a(y(aw))$ contains three occurrences of a. Since a channel can both send and receive, and in particular, usually, first sends and then receives, we are led to choose again the rightmost occurrence of a channel application as the receiving one. Nonetheless, since the threads of a single process do not communicate with each other, it is best to let all of them receive the message in correspondence of their *locally rightmost* channel application. Thus in the example, both $a\,u$ and $a\,w$ will receive messages.

The third choice is what to do with the arguments of a receiving channel. After a few programming examples, like the Floyd–Warshall algorithm of Section 5.6, it is natural to convince oneself that it is better to keep the arguments, a feature that we call *memory*. In the previous term, if $a\,u$ and $a\,w$ receive s, they will be replaced respectively, with $\langle u, s\rangle$ and $\langle w, s\rangle$.

Continuing our example and summing up, we will have the following reduction:

$$(*) \qquad \mapsto_{\mathtt{p}} \qquad {}_a(\ (a\,r \parallel a(x(a\,s)))\ \parallel\ (\langle u, s\rangle \parallel a(y\langle w, s\rangle))\ )$$

As we can see, the message s in correspondence of the rightmost occurrence of a in $a(x(a\,s))$ is transmitted by the first process to the rightmost application of a in each one of the two threads $a\,u$ and $a(y(aw))$ of the second process.

The fourth choice to make is which processes should receive messages and which processes should send them. As anticipated in the previous section, we are guided by the typing. Let us consider for instance the term

$$ {}_a(\ x\,(a^{A\to A\wedge B}\,s)\ \|\ \ y\,(a^{B\to B\wedge A}\,t)\ ) $$

where s and t are specific simply typed λ-terms, $x : A \wedge B \to C$ and $y : B \wedge A \to C$. A mini cross reduction rule corresponding to this typing rule admits communication in two directions, from left to right and from right to left, because looking at the types, the second process can receive messages from the first and viceversa. The reduction rule for the left to right direction, for instance, is

$$ {}_a(\ x\,(a^{A\to A\wedge B}\,s)\ \|\ \ y\,(a^{B\to B\wedge A}\,t)\ )\ \mapsto_{\mathtt{p}}\ {}_a(\ x\,(a^{A\to A\wedge B}\,s)\ \|\ \ y\,\langle t, s\rangle\ ) $$

whereas from right to left is

$$ {}_a(\ x\,(a^{A\to A\wedge B}\,s)\ \|\ \ y\,(a^{B\to B\wedge A}\,t)\ )\ \mapsto_{\mathtt{p}}\ {}_a(\ x\,\langle s, t\rangle\ \|\ y\,(a^{B\to B\wedge A}\,t)\ \ ) $$

We can thus see that our reduction rules are naturally non-deterministic. When writing actual code (Section 5.6), however, we shall fix an intuitive reduction strategy that will choose deterministically at each step which process should receive the next batch of messages. This way programmers can predict the behavior of their parallel code.

The fifth, and last, choice to make is what to do with the threads or processes that do not contain any communication channel. The idea is that whenever a term contains no channel occurrence, it has already reached a result, as it does not need to interact with the context further. Hence at the end of the computation we may wish to select out some of the processes that have reached their own results and consider them all together the global result of the computation. Thus we introduce the simplification reduction (see Table 5.2).

To precisely define communication reductions, we need to introduce two kinds of contexts: one for terms that can communicate, one for terms which are in parallel but cannot communicate.

Definition 5.2.1 (Simple Parallel Term)**.** A **simple parallel term** is a $\lambda_{\|}$-term $t_1 \| \ldots \| t_n$, where each t_i, for $1 \leq i \leq n$, is a simply typed λ-term.

Definition 5.2.2. A **context** $\mathcal{C}[\ ]$ is a $\lambda_{\|}$-term with some fixed variable $[\ ]$ occurring exactly once.

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

For any $\lambda_{\|}$-term u of the same type of $[\ ]$, $\mathcal{C}[u]$ denotes the term obtained replacing $[\ ]$ with u in $\mathcal{C}[\ ]$, *without renaming bound variables.*

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

Communication Reductions

Mini cross reductions

$$\begin{array}{c} {}^{\mathcal{A}}_{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots) \\ \mapsto_{\mathtt{p}} \\ {}^{\mathcal{A}}_{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[\langle v_1, w_1, \ldots, w_p\rangle] \parallel \ldots \parallel \mathcal{D}_n[\langle v_n, w_1, \ldots, w_p\rangle] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots) \end{array}$$

where $\mathcal{C}_1[a\, w_1], \ldots, \mathcal{C}_p[a\, w_p]$ are all the processes outlinked to the process $(\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots)$; each $\mathcal{D}_j$ is a simple context and each $\mathcal{C}_j$ is a simple parallel context; the displayed occurrences of a are rightmost in each $\mathcal{D}_j[a\, v_j]$ and in each $\mathcal{C}_j[a\, w_j]$; finally, the free variables of each w_j are free in $\mathcal{C}_j[a\, w_j]$.

Simplification reductions

$${}^{\mathcal{A}}_{a}((u_1 \parallel \ldots \parallel u_n) \parallel \ldots \parallel (u_m \parallel \ldots \parallel u_p)) \mapsto_{\mathtt{p}} u_{i_1} \parallel \ldots \parallel u_{i_q}$$

whenever $u_{i_1}, \ldots, u_{i_q}$ do not contain a and $1 \leq i_1 < \ldots < i_q \leq p$.

Table 5.2: Reduction Rules for $\lambda_{\parallel}$.

The reduction rules of $\lambda_{\parallel}$ are presented in Table 5.2. As usual, we adopt the reduction schema: $\mathcal{C}[t] \mapsto_{\mathtt{p}} \mathcal{C}[u]$ whenever $t \mapsto_{\mathtt{p}} u$ and for any context $\mathcal{C}$. With $\mapsto^*_{\mathtt{p}}$ we shall denote the reflexive and transitive closure of the one-step reduction $\mapsto_{\mathtt{p}}$.

We explain now the general case of the mini cross reduction. The rule identifies a single process as the receiver and, as consequence, possibly several processes as senders. Once the receiving process is fixed, the senders are determined by the axiom schema $\mathcal{A}$ that the type of the communication channel occurring in the receiving process instantiates. In particular, in the term

$$(\star) \quad {}^{\mathcal{A}}_{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots)$$

the processes $\mathcal{C}_1[aw_1\ ], \ldots, \mathcal{C}_p[aw_p\ ]$ are the senders and the whole process

$$(\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots)$$

is the receiver; formally, $\mathcal{C}_1[aw_1\ ], \ldots, \mathcal{C}_p[aw_p\ ]$ are all the process *outlinked* (see Definition 5.1.2) to $(\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots)$. In this latter process, we have highlighted the threads that actually contain the channel a: all of them will receive the messages. Consistently with our choices, the displayed occurrences of a are rightmost in each $\mathcal{D}_j[a\, v_j]$ and in each $\mathcal{C}_j[a\, w_j]$ The processes containing $w_1, \ldots, w_p$ send them to all the rightmost occurrences of a in the processes $\mathcal{D}_1, \ldots, \mathcal{D}_n$: hence $(\star) \quad \mapsto_{\mathtt{p}}$

$${}^{\mathcal{A}}_{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[\langle v_1, w_1, \ldots, w_p\rangle] \parallel \ldots \parallel \mathcal{D}_n[\langle v_n, w_1, \ldots, w_p\rangle] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots)$$

provided that for each w_j, its free variables are free in $\mathcal{C}_j[a\, w_j]$: this condition is needed to avoid that bound variables become free, violating the Subject Reduction. Whenever w_j is a closed term – executable code – the condition is automatically satisfied. As we can see, the reduction retains all of the terms $v_1, \ldots, v_n$ occurring in $\mathcal{D}_1, \ldots, \mathcal{D}_n$ before the communication.

Notice that a process can receive a batch of messages only if all the processes outlinked to it are ready to communicate. Which in turn implies that no incomplete batches of messages are received: every transmission that a process receives will contain one message from each process that can transmit to the first process. This is quite natural from a logical point of view and greatly simplifies the communication reductions. From a computational point of view, this behavior enforces a style of programming that we could call receiver-oriented. In other terms, a sender does not actively send the message, it just makes the message available. A receiver, on the other hand, can actively collect a batch of messages when a complete one is made available from the relative senders.

Remark 5.2 (Using channels and consuming them)**.** Unlike in π-calculus [Mil92, SW03], in $\lambda_{\|}$ the channel that sends the message is not consumed. The reason is precisely that the mini cross reduction rule adopts the perspective of the receiver rather than that of the senders. A process may wish to send the same message to multiple processes, not only to the current receiver, so it is reasonable to let that possibility open and keep the same output channel application active after the reduction. Therefore, a process will stop sending the same message only when its output channel will be turned into an input channel, that is, when the process will be selected as current receiver. Again, when coding (Section 5.6), we shall introduce a discipline that regulates how processes send messages to their potential receivers.

We show now that $\lambda_{\|}$ computation steps preserve the type of terms.

Theorem 5.2.1 (Subject Reduction)**.** *If $t : A$ and $t \mapsto_{\mathtt{p}} u$, then $u : A$ and all the free variables of u appear among those of t.*

Proof. See Theorem 2.4.3 for the case of intuitionistic reductions. Since showing the statement for simplification reductions is trivial, we only consider the case of mini cross reductions.

Reductions of the form $\overset{\mathcal{A}}{a}((u_1 \parallel \ldots \parallel u_n) \parallel \ldots \parallel (u_m \parallel \ldots \parallel u_p)) \mapsto_{\mathtt{p}} u_{i_1} \parallel \ldots \parallel u_{i_q}$ trivially preserve the type of terms since all processes u_k must have the same type. It is clear, moreover, that no new free variable is created.

Suppose then that we perform a mini cross reduction step as follows:

$$\begin{array}{c} \overset{\mathcal{A}}{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[a^{A_i \to A_i \wedge B_i}\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a^{A_i \to A_i \wedge B_i}\, v_n] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots) \\ \mapsto_{\mathtt{p}} \\ \overset{\mathcal{A}}{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[\langle v_1, w_1, \ldots, w_p\rangle] \parallel \ldots \parallel \mathcal{D}_n[\langle v_n, w_1, \ldots, w_p\rangle] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots) \end{array}$$

Since the type of the channel a is an instance $(A_1 \to A_1 \wedge B_1) \vee \ldots \vee (A_m \to A_m \wedge B_m)$ of the schema $\mathcal{A} = (A_1 \to A_1 \wedge B_1) \vee \ldots \vee (A_m \to A_m \wedge B_m)$ where $B_i = A_{k_1} \wedge \ldots \wedge A_{k_p}$, and since $w_1 : A_{k_1}, \ldots, w_p : A_{k_p}$, then the term $(\ldots \parallel \mathcal{D}_1[\langle v_1, w_1, \ldots, w_p\rangle] \parallel \ldots \parallel \mathcal{D}_n[\langle v_n, w_1, \ldots, w_p\rangle] \parallel \ldots)$ is well defined and its type is the same as that of $(\ldots \parallel \mathcal{D}_1[a^{A_i \to A_i \wedge B_i}\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a^{A_i \to A_i \wedge B_i}\, v_n] \parallel \ldots)$. Hence the type of the term does not change.

Since the displayed occurrences of a are rightmost in each $\mathcal{C}_j[a\,w_j]$ and thus a does not occur in w_j, no occurrence of a with type different from $A_i \to A_i \wedge B_i$ occurs in the terms $\mathcal{D}_1[\langle v_1, w_1, \ldots, w_p\rangle], \ldots, \mathcal{D}_n[\langle v_n, w_1, \ldots, w_p\rangle]$. Finally, the free variables of each w_j are free also in $\mathcal{C}_j[a\,w_j]$, and thus no new free variable is created by the reduction. □

The termination of $\lambda_{\|}$ programs is established in Section 5.5.

5.3 Properties of the communication in $\lambda_{\|}$

Mini cross reductions do not have any communication mechanism for sending terms outside their computational environment. Communications of this kind can be carried out, for example, in λ_{Cl} and λ_{G}, but not in $\lambda_{\|}$. In particular, the configurations that block communication in $\lambda_{\|}$ are of the form $\mathcal{C}[au]$ where $\mathcal{C}[au]$ is a simply typed λ-term and u contains some free variables which are bound in $\mathcal{C}[au]$. We define here a class of terms that do not contain these configurations and that reduce only into similar terms: the class of *communicative* terms.

Definition 5.3.1. We say that a term t is *communicative* if it satisfies the following conditions:

1. for any channel a, if a term au occurs in t, then no intuitionistic variable is free in u
2. all channel variables occurring in t are applied to some term

Condition 1. implies that t does not contain simply typed subterms $\mathcal{C}[au]$ where u contains some free variables which are bound in $\mathcal{C}[au]$. Condition 2. is needed in order to prove that reductions do not transform a communicative term into a term which is not communicative.

There are terms which are not communicative but still reduce into communicative terms. An interesting example is the following term: ${}_a((\lambda x\,\lambda y\,y)(\lambda z\,a(z))(\lambda z\,z) \parallel a(\lambda x\,x))$ which reduces to ${}_a(\lambda z\,z \parallel a(\lambda x\,x))$ and then can be simplified into $\lambda z\,z$. Not only this term is not communicative, but it actually contains the configuration $\lambda z\,a(z)$ that blocks the communications through $a(z)$. The term ${}_a(\lambda z\,z \parallel a(\lambda x\,x))$ into which the first reduces, instead, is clearly communicative as we can see in the proof of Proposition 5.3.1 below.

We show now that the class of communicative terms is closed under reductions. In the proof we use the following definitions.

Definition 5.3.2 (Disjoint occurrences)**.** For any term u, two occurrences s and t of subterms of u are disjoint if s does not occur in t and t does not occur in s.

Definition 5.3.3 (I-closed)**.** A term u is *i-closed* if no intuitionistic variable is free in u.

The intuitions behind the next proof are quite simple. All applied channels remain applied also after reductions because no reduction can transform a channel application au into a pair $\langle a, \ldots\rangle$ or $\langle \ldots, a\rangle$, into an abstraction $\lambda y\, a$, and certainly does not transform the term into a single variable occurrence a. There are three main reasons why channel applications au in which u is i-closed remain terms of this kind during reduction: first, reductions do not transform bound variables into free variables; second, the messages which are transmitted inside a communicative term are all closed terms; and third, by reducing an application we do not modify i-closed terms occurring in the body of the applied term.

Proposition 5.3.1. *If t is communicative and $t \mapsto_{\mathtt{p}} t'$, then t' is communicative.*

Proof. We assume that t is communicative and we prove that t' is communicative. We reason by induction on the size of t and distinguish four cases corresponding to the reduction applied to t.

- Suppose that the reduction is $(\lambda x\, v)w \mapsto_{\mathtt{p}} v[w/x]$. If a non-applied channel occurs in t', then it either occurs in w or in v or in a subterm which is disjoint with respect to $v[w/x]$. In all cases, the channel also occurs in t as a term which is not applied, which contradicts the assumption that t is communicative.

  We show now that for any subterm au of t', u is i-closed. Consider any occurrence au of a subterm of t'. If au and $v[w/x]$ are disjoint in t', then au and $(\lambda x\, v)w$ are disjoint in t and hence u must be i-closed, otherwise t is not communicative. Suppose then that au and $v[w/x]$ are not disjoint. We have two cases.

  1. au occurs in $v[w/x]$. Since either a occurs in w or in v, then au either occurs in w or $au = au'[w/x]$ occurs in $v[w/x]$. Since no term au' such that x is free in u' can occur in v, $u = u'[w/x] = u'$ and au occurs in v. Hence, in both cases au occurs also in t and u must be a i-closed term.
  2. $v[w/x]$ occurs in u. Let us denote by u' the term from which we obtain u by reducing $(\lambda x\, v)w \mapsto_{\mathtt{p}} v[w/x]$. The term u' must be i-closed because otherwise t would not be communicative. Since, moreover, the reduction $(\lambda x\, v)w \mapsto_{\mathtt{p}} v[w/x]$ does not generate any free variables in u', we have that u is a i-closed term as well and we are done.

- Suppose that the reduction is $\langle v_0, v_1\rangle \pi_i \mapsto_{\mathtt{p}} v_i$. If a non-applied channel occurs in t', then it either occurs in v_i or in a subterm which is disjoint with respect to v_i. In both cases, it also occurs in t as a term which is not applied, which contradicts the assumption that t is communicative.

  We show now that for any subterm au of t', u is i-closed. Consider now a generic occurrence au of a subterm of t'. If au and v_i are disjoint in t', then au and $\langle v_0, v_1\rangle$ are disjoint in t and hence u must be i-closed, otherwise t is not communicative. Suppose then that au and v_i are not disjoint. We have two cases.

1. au occurs in v_i. Hence au occurs also in t and u is i-closed because otherwise t would not be communicative.
2. v_i occurs in au. Since $v_i = au$ falls under the previous case, we only consider the case in which v_i occurs in u. Let us denote by u' the term from which we obtain u by reducing $\langle v_0, v_1\rangle\pi_i \mapsto_{\mathtt{p}} v_i$. The term u' must be i-closed because otherwise t would not be communicative. Since, moreover, the reduction $\langle v_0, v_1\rangle\pi_i \mapsto_{\mathtt{p}} v_i$ does not generate any free variables in u', we have that u is a i-closed term as well and we are done.

- Suppose that the reduction is

$$t \quad = \quad {}_a(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots)$$

$$\mapsto_{\mathtt{p}}$$

$${}_a(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[\langle v_1, w_1, \ldots, w_p\rangle] \parallel \ldots \parallel \mathcal{D}_n[\langle v_n, w_1, \ldots, w_p\rangle] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots)$$

  If a non-applied channel occurs in t', then it occurs in one of the terms $w_1, \ldots, w_p$ or in a term v_j for some $j \in \{1, \ldots, n\}$ or in a subterm which is disjoint with respect to $\langle v_j, w_1, \ldots, w_p\rangle$ for any $j \in \{1, \ldots, n\}$. In all cases, the channel also occurs in t as a term which is not applied, which contradicts the assumption that t is communicative.

  We show now that for any subterm au of t', u is i-closed. Consider now any occurrence au of a subterm of any $\mathcal{D}_j[\langle v_j, w_1, \ldots, w_p\rangle]$. If au and $\langle v_j, w_1, \ldots, w_p\rangle$ are disjoint in t', then au and $a\, v_j$ are disjoint in t and hence u must be i-closed, otherwise t is not communicative. Suppose then that au and $\langle v_j, w_1, \ldots, w_p\rangle$ are not disjoint. We have two cases.

  1. au occurs in $\langle v_j, w_1, \ldots, w_p\rangle$. Hence au must occur in one of the terms $v_j, w_1, \ldots, w_p$ and therefore in t. Then the term u is i-closed, because otherwise t would not be communicative.
  2. $\langle v_j, w_1, \ldots, w_p\rangle$ occurs in au. Since $\langle v_j, w_1, \ldots, w_p\rangle \neq au$ we know that $\langle v_j, w_1, \ldots, w_p\rangle$ occurs in u. Let us denote by u' the term from which we obtain u by replacing $a\, v_j$ with $\langle v_0, v_1\rangle\pi_i \mapsto_{\mathtt{p}} v_i$. The term u' and the terms $v_j, w_1, \ldots, w_p$ must be i-closed because otherwise t would not be communicative. Hence u is a i-closed term as well and we are done.

- Suppose that the reduction is

$$\overset{\mathcal{A}}{{}_a}((u_1 \parallel \ldots \parallel u_n) \parallel \ldots \parallel (u_m \parallel \ldots \parallel u_p)) \mapsto_{\mathtt{p}} u_{i_1} \parallel \ldots \parallel u_{i_q}$$

  Since all simply typed λ-terms that occur in t' also occur in t, t' is communicative because otherwise t would not be communicative.

□

In conclusion we have that, during the normalization of a closed communicative term, all channels are either used as receivers in a communication, and hence consumed, or discarded. Therefore, no communication deadlock can occur in a communicative term.

Corollary 5.3.1.1 (Closed communicative terms are communication-deadlock-free)**.** *For any term t, if t is closed, communicative and normal, then t does not contain any communication channel.*

Proof. Suppose for the sake of contradiction that t is closed, communicative and normal, but t contains a communication channel. Since t is closed and contains a communication channel, t is of the form

$$\overset{\mathcal{A}}{a}((u_1 \parallel \ldots \parallel u_n) \parallel \ldots \parallel (u_m \parallel \ldots \parallel u_p))$$

We have then two cases.

1. All terms $u_1, \ldots, u_n, \ldots, u_m, \ldots, u_p$ contain communication channels. Since t is communicative, all channels occurring in t are applied to i-closed terms. Therefore, we can rewrite t as $\overset{\mathcal{A}}{a}(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots)$ where $\mathcal{C}_1[a\, w_1], \ldots, \mathcal{C}_p[a\, w_p]$ are all the processes outlinked to the process $(\ldots \parallel \mathcal{D}_1[a\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a\, v_n] \parallel \ldots)$; each $\mathcal{D}_j$ is a simple context and each $\mathcal{C}_j$ is a simple parallel context; the displayed occurrences of a are rightmost in each $\mathcal{D}_j[a\, v_j]$ and in each $\mathcal{C}_j[a\, w_j]$; and the free variables of each w_j are free in $\mathcal{C}_j[a\, w_j]$. But this means that a mini cross reduction can be applied, which contradicts the assumption that t is normal.

2. Some of the terms $u_1, \ldots, u_n, \ldots, u_m, \ldots, u_p$ does not contain communication channels. This contradicts the assumption that t is normal since a simplification reduction can be applied.

□

5.4 From communication topologies to programs

We present now a method for automatically extracting $\lambda_{\parallel}$ typing rules from graph-specified topologies. Namely, given a direct, reflexive graph G whose nodes and edges represent respectively processes and communication channels, we describe how to transform it into a formula $\mathcal{A} \in \mathrm{Ax}$ corresponding to a typing rule $(\mathcal{A})$ for $\lambda_{\parallel}$-terms, see Table 5.3. We will show that this typing rule encodes a topology which exactly mirrors the graph G: two processes may comunicate if and only the corresponding graph nodes are connected by an edge and the direction of communication follows the edge.

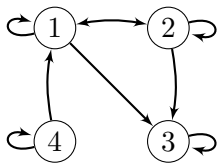
Before establishing the exact correspondence between directed graphs and $\lambda_{\parallel}$ communication reductions, we show a simple example.

Given a directed reflexive graph $G = (V, E)$, the axiom schema $\mathcal{A}$ encoding G is a disjunction $C_1 \vee \ldots \vee C_k$ such that k is the cardinality of V and for each $n \in \{1, \ldots, k\}$:

- $C_n = A_n \to A_{i_1} \wedge \ldots \wedge A_{i_m}$, if the nth node in G has incoming edges from more than one node among $i_1, \ldots, i_m$
- $C_n = A_n \to A_n \wedge \bot$, if the n-th node in G has only one incoming edge

Table 5.3: Procedure to extract an axiom $\mathcal{A} \in \mathrm{Ax}$ from a directed reflexive graph G.

Example 5.4.1. Consider the graph



The axiom extracted by the procedure in Table 5.3 is the following

$$(A_1 \to A_1 \wedge A_2 \wedge A_4) \vee (A_2 \to A_2 \wedge A_1) \vee (A_3 \to A_3 \wedge A_1 \wedge A_2) \vee (A_4 \to A_4 \wedge \bot)$$

The relative typing rule is the following, where $u_i \in \{u_1, \ldots, u_4\}$ stands for the i-th node of the graph

$$\frac{\begin{array}{cccc}[a : A_1 \to A_1 \wedge A_2 \wedge A_4] & [a : A_2 \to A_2 \wedge A_1] & [a : A_3 \to A_3 \wedge A_1 \wedge A_2] & [a : A_4 \to A_4 \wedge \bot] \\ \vdots & \vdots & \vdots & \vdots \\ u_1 : F & u_2 : F & u_3 : F & u_4 : F\end{array}}{{}_a(u_1 \parallel u_2 \parallel u_3 \parallel u_4) : F}$$

As defined in Table 5.2 the mini cross reductions for the terms typed by this rule are

$${}_a(\mathcal{C}_1[a\,t_1] \parallel \mathcal{C}_2[a\,t_2] \parallel \mathcal{C}_3 \parallel \mathcal{C}_4[a\,t_4]) \quad \mathcal{C}_1 \text{ receives } \mapsto_{\mathtt{p}} \quad {}_a(\mathcal{C}_1[\langle t_1, t_2, t_4\rangle] \parallel \mathcal{C}_2[a\,t_2] \parallel \mathcal{C}_3 \parallel \mathcal{C}_4[a\,t_4])$$

$${}_a(\mathcal{C}_1[a\,t_1] \parallel \mathcal{C}_2[a\,t_2] \parallel \mathcal{C}_3 \parallel \mathcal{C}_4) \quad \mathcal{C}_2 \text{ receives } \mapsto_{\mathtt{p}} \quad {}_a(\mathcal{C}_1[a\,t_1] \parallel \mathcal{C}_2[\langle t_2, t_1\rangle] \parallel \mathcal{C}_3 \parallel \mathcal{C}_4)$$

$${}_a(\mathcal{C}_1[a\,t_1] \parallel \mathcal{C}_2[a\,t_2] \parallel \mathcal{C}_3[a\,t_3] \parallel \mathcal{C}_4) \quad \mathcal{C}_3 \text{ receives } \mapsto_{\mathtt{p}} \quad {}_a(\mathcal{C}_1[a\,t_1] \parallel \mathcal{C}_2[a\,t_2] \parallel \mathcal{C}_3[\langle t_3, t_1, t_2\rangle] \parallel \mathcal{C}_4)$$

These reductions correspond to the edges of the graph: u_1 and u_2 can exchange messages in both directions, both u_1 and u_2 can send messages to u_3, which can only receive messages but cannot transmit, and finally u_4 can send messages to u_1 but cannot receive any message. Moreover, as discussed in Section 5.2, the occurrence of A_i in the consequent of each implication provides us with a memorization mechanism for messages, corresponding to the reflexive edges of the graph.

We prove now that the rule extracted from a graph constrains communications to happen as indicated by the edges of the graph.

Proposition 5.4.1 (Topology correspondence)**.** *For any directed reflexive graph G and term ${}_a(u_1 \parallel \ldots \parallel u_m)$ typed using the rule corresponding to the axiom extracted from G by the procedure in Table 5.3, there exists a basic cross reduction for ${}_a(u_1 \parallel \ldots \parallel u_m)$ that transmits a term w from u_x to u_y if and only if G contains an edge from x to y.*

Proof. Suppose that G contains an edge from x to y. The procedure in Table 5.3 generates an axiom containing a disjunct of the form $A_y \to A_y \wedge \ldots \wedge A_x \wedge \ldots \wedge A_z$ and a disjunct of the form $A_x \to (A_x \wedge \ldots)$. We thus have a reduction of the form

$$\begin{aligned} &{}_a(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\mathcal{D}_1[a^{A_i \to A_i \wedge B_i}\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a^{A_i \to A_i \wedge B_i}\, v_n]) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots) \\ &\mapsto_{\mathtt{p}} {}_a(\ldots \parallel \mathcal{C}_1[aw_1] \parallel \ldots \parallel (\ldots \parallel \mathcal{D}_i[\langle v_1, w_1, \ldots, w_x, \ldots, w_p\rangle] \parallel \ldots) \parallel \ldots \parallel \mathcal{C}_p[aw_p] \parallel \ldots) \end{aligned} \tag{5.1}$$

for $u_x = \mathcal{C}_x[a\, w_x] =$ and $u_y = \mathcal{D}_1[a^{A_i \to A_i \wedge B_i}\, v_1] \parallel \ldots \parallel \mathcal{D}_n[a^{A_i \to A_i \wedge B_i}\, v_n]$. Such reductions allow the term w_x to be transmitted from the term u_x to the reduced form

$$(\ldots \parallel \mathcal{D}_i[\langle v_1, w_1, \ldots, w_x, \ldots, w_p\rangle] \parallel \ldots)$$

of the term u_y.

As for the other direction, due to the conditions of basic cross reductions in Table 5.2, reductions of the form (5.1) for a term $w_x : A_x$ directly require that the axiom contains a disjunct of the form $A_x \to (\ldots)$ and a disjunct of the form $A_y \to (A_y \wedge \ldots \wedge A_x \wedge \ldots \wedge A_z)$. But the procedure in Table 5.3 produces such disjoints only if the graph G contains nodes x, y and an edge from x to y. □

5.5 The strong normalization theorem

We prove the strong normalization theorem for $\lambda_{\parallel}$: any reduction of every $\lambda_{\parallel}$-term ends in a finite number of steps into a normal form. This means that the computation of any typed $\lambda_{\parallel}$-term always terminates independently of the chosen reduction strategy. The strong normalization argument for $\lambda_{\parallel}$ is new. We reduce the strong normalization of $\lambda_{\parallel}$ to the strong normalization of a non-deterministic reduction relation over simply typed λ-terms. The idea is to simulate communication by non-determinism, a technique inspired by [AZ16].

Definition 5.5.1 (Normal forms and strongly normalizable terms)**.** A $\lambda_{\parallel}$-term u of $\lambda_{\parallel}$ is *strongly normalizable* if every reduction of u is finite. With $\mathrm{SN}_{\mathtt{p}}$ we denote the set of the strongly normalizable terms of $\lambda_{\parallel}$.

5.5.1 Non-deterministic reductions

The strong normalization problem for parallel λ-calculi is an intricate one. Decreasing complexity measures are quite hard to come up with and indeed those introduced for

λ_{Cl} and λ_{G} fail to show that fragments of $\lambda_{\|}$ are strongly normalizing. Given a term ${}_a(u_1 \parallel \ldots \parallel u_n)$, one would like to measure its complexity as a function of the complexities of the terms u_i, for example taking into account the number of channel occurrences and the length of the longest reduction of λ-redexes in u_i. However, when u_i receives a message its code may drastically change and both those numbers may increase. Moreover, there is a potential circularity to address: channels send messages, so they may generate new λ-calculus redexes in the receivers; in turn, λ-calculus redexes may duplicate channel occurrences, generating even more communications and messages.

In order to break this circularity, the idea is to use a radically different complexity measure. The complexity of a process u_i should take into account all the possible messages that u_i may receive. Now, if $u_i = \mathcal{D}[a\,t]$, then after receiving a message it becomes $\mathcal{D}[\langle t, s\rangle]$, where s is an *arbitrary*, from the point of view of u_i, simply typed λ-term. We thus create a reduction relation over simply typed λ-terms that simulates the reception "out of the blue" of this kind of messages. Namely, we extend the reduction relation of λ-calculus with the rule $a^T \rightsquigarrow t$, for every channel a^T and simply typed λ-term $t : T$ that does not contain channels. In order to simulate the reception of an arbitrary batch w of messages, the simply typed λ-term $t : T$ will be instantiated as $\lambda x\,\langle x, w\rangle$ in the proof of Theorem 5.5.6. With this reductions, λ-terms not containing channels are the usual deterministic ones.

Definition 5.5.2 (Deterministic simply typed λ-terms)**.** A simply typed λ-term t is called *determistic*, if t does not contain any channel occurrence.

Definition 5.5.3 (The non-deterministic reduction relation $\rightsquigarrow$)**.** The reduction relation $\rightsquigarrow$ over simply typed λ-terms is defined as extension of the relation $\mapsto_{\mathtt{p}}$ as follows:

$$(\lambda x^A\,u)\,t \rightsquigarrow u[t/x^A]$$

$$\langle u_0, u_1\rangle\,\pi_i \rightsquigarrow u_i, \text{ for } i = 0, 1$$

$$a^T \rightsquigarrow t, \quad \text{for every channel } a^T \text{ and deterministic simply typed λ-term } t : T$$

and as usual we close by contexts: $\mathcal{C}[t] \rightsquigarrow \mathcal{C}[u]$ whenever $t \rightsquigarrow u$ and $\mathcal{C}[\,]$ is a simple context. With $\rightsquigarrow^*$ we shall denote the reflexive and transitive closure of the one-step reduction $\rightsquigarrow$.

The plan of our proof will be to prove the strong normalization of simply typed λ-calculus with respect to the reduction $\rightsquigarrow$ (Corollary 5.5.5.1) and then derive the strong normalization of $\lambda_{\|}$ using $\rightsquigarrow$ as the source of the complexity measure (Theorem 5.5.5). We shall prove the first result by the standard Tait-Girard reducibility technique (Definition 5.5.4).

In the following, we define SNN to be the set of strongly normalizing simply typed λ-terms with respect to the non-deterministic reduction $\rightsquigarrow$. The reduction tree of a strongly normalizable term with respect to $\rightsquigarrow$ is no more finite, but still well-founded. It is well-known that it is possible to assign to each node of a well-founded tree an ordinal number, in such a way that it decreases passing from a node to any of its children. We

will call the *ordinal size* of a term $t \in \mathrm{SNN}$ the ordinal number assigned to the root of its reduction tree and we denote it by $h(t)$; thus, if $t \rightsquigarrow u$, then $h(t) > h(u)$. To fix ideas, one may define $h(t) := \mathsf{sup}\{h(u) + 1 \mid t \mapsto_{\mathtt{p}} u\}$.

5.5.2 Reducibility and properties of reducible terms

We define a notion of reducibility for simply typed λ-terms with respect to the reduction $\rightsquigarrow$. As usual, we shall prove that every reducible term is strongly normalizable and afterwards that all simply typed λ-terms are reducible. The difference with the usual reducibility proof is that we shall first prove that deterministic simply typed λ-terms are reducible (Theorem 5.5.3), which makes it possible to prove that channels are reducible (Proposition 5.5.4) and finally that all terms are reducible (Theorem 5.5.5). This amounts to prove twice the usual Adequacy Theorem, normally only proved as the final result. We prove it here once as Theorem 5.5.3 and once as Theorem 5.5.5.

Definition 5.5.4 (Reducibility)**.** Assume $t : C$ is a simply typed λ-term. We define the relation $t\,\mathtt{r}\,C$ ("t is reducible of type C") by induction and by cases according to the form of C:

1. $t\,\mathtt{r}\,\mathsf{P}$, with P atomic, if and only if $t \in \mathrm{SNN}$
2. $t\,\mathtt{r}\,A \wedge B$ if and only if $t\,\pi_0\,\mathtt{r}\,A$ and $t\,\pi_1\,\mathtt{r}\,B$
3. $t\,\mathtt{r}\,A \to B$ if and only if for all u, if $u\,\mathtt{r}\,A$, then $tu\,\mathtt{r}\,B$

We prove that the set of reducible terms for a given formula C satisfies the usual properties of a Girard reducibility candidate. Following [GLT89], neutral terms are terms that did not reach a stable form but need further computation.

Definition 5.5.5 (Neutrality)**.** A term is neutral if it is not of the form $\lambda x\,u$ or $\langle u, t\rangle$.

Definition 5.5.6 (Reducibility Candidates)**.** Extending the approach of [GLT89], we define four properties **(CR1)**, **(CR2)**, **(CR3)** of reducible terms t:

(CR1) If $t\,\mathtt{r}\,A$, then $t \in \mathrm{SNN}$.

(CR2) If $t\,\mathtt{r}\,A$ and $t \rightsquigarrow^* t'$, then $t'\,\mathtt{r}\,A$.

(CR3) If t is neutral and for every t', $t \rightsquigarrow t'$ implies $t'\,\mathtt{r}\,A$, then $t\,\mathtt{r}\,A$.

We show that every term t possesses the reducibility candidate properties. The arguments for **(CR1)**, **(CR2)**, **(CR3)**, are in many cases standard (see [GLT89]).

Proposition 5.5.1. *Let $t : C$ be a simply typed λ-term. Then t has the properties **(CR1)**, **(CR2)**, **(CR3)**.*

Proof. By induction on C. The base case is when C is atomic. Then $t \mathtt{r} C$ means $t \in \mathrm{SNN}$. Therefore **(CR1)**, **(CR2)**, **(CR3)** are trivial. Assume that $C = A \to B$ (the case $C = A \wedge B$ is similar).

(CR1). Suppose $t \mathtt{r} A \to B$. By i.h. **(CR3)**, for any intuitionistic variable x, we have $x \mathtt{r} A$. Therefore, $t\, x \mathtt{r} B$, and by **(CR1)**, $t\, x \in \mathrm{SNN}$, and thus $t \in \mathrm{SNN}$.

(CR2). Suppose $t \mathtt{r} A \to B$ and $t \rightsquigarrow t'$. Let $u \mathtt{r} A$: we have to show $t'u \mathtt{r} B$. Since $tu \mathtt{r} B$ and $tu \rightsquigarrow t'u$, the i.h. leads to **(CR2)** that $t'u \mathtt{r} B$.

(CR3). Assume t is neutral and $t \rightsquigarrow t'$ implies $t' \mathtt{r} A \to B$. Suppose $u \mathtt{r} A$; we have to show that $tu \mathtt{r} B$. We proceed by induction on $h(u)$ since $u \in \mathrm{SNN}$ by i.h. **(CR1)**. By i.h. **(CR3)** holds for the type B. So assume $tu \rightsquigarrow z$; it is enough to show that $z \mathtt{r} B$. If $z = t'u$, with $t \rightsquigarrow t'$, then by hypothesis $t' \mathtt{r} A \to B$, so $z \mathtt{r} B$. If $z = tu'$, with $u \rightsquigarrow u'$, by i.h. **(CR2)** $u' \mathtt{r} A$, and therefore $z \mathtt{r} B$ by the i.h. relative to the size of the reduction tree of u'. There are no other cases since t is neutral. □

The next task is to prove that all introduction rules of simply typed λ-calculus define a reducible term from a list of reducible terms for all premises.

In some cases that is true by definition of reducibility; we list below some non-trivial but standard cases we have to prove.

Proposition 5.5.2.

1. *If for every $t \mathtt{r} A$, $u[t/x] \mathtt{r} B$, then $\lambda x\, u \mathtt{r} A \to B$.*
2. *If $u \mathtt{r} A$ and $v \mathtt{r} B$, then $\langle u, v \rangle \mathtt{r} A \wedge B$.*

Proof.

1. We fix now a generic s such that $s \mathtt{r} A$ and we show that if for every t such that $t \mathtt{r} A$, $u[t/x] \mathtt{r} B$ holds, then the term resulting from $(\lambda x\, u)s$ after one step of reduction is reducible. Since **(CR3)** implies that $x \mathtt{r} A$, we have that $u \mathtt{r} B$. Hence, since both $s \mathtt{r} A$ and $u \mathtt{r} B$, by **(CR1)** we can reason by induction on the lexicographical order on the ordered pairs $(h(u), h(s))$ of the ordinal sizes of u and s. We consider the possible forms of the term resulting from the reduction:
   - $u[s/x]$. By assumption, since $s \mathtt{r} A$, $u[s/x] \mathtt{r} B$ holds and hence we are done.
   - $(\lambda x\, u')s$. Since for every t such that $t \mathtt{r} A$ it holds that $u[t/x] \mathtt{r} B$ and $u[t/x] \rightsquigarrow^* u'[t/x]$, by **(CR2)** we have that $u'[t/x] \mathtt{r} B$. Since $u'[t/x] \mathtt{r} B$ holds for any t such that $t \mathtt{r} A$ and since $h(u') < h(u)$, the thesis follows by induction hypothesis.
   - $(\lambda x\, u)s'$. By **(CR2)** and since $s \mathtt{r} A$, we have that $s' \mathtt{r} A$. Since, by assumption, for every t such that $t \mathtt{r} A$, $u[t/x] \mathtt{r} B$ holds and since $h(s') < h(s)$, the thesis follows by induction hypothesis.

In all cases, we obtain a reducible term. By **(CR3)** we can conclude that $(\lambda x\, u)s$ is reducible as well. Since moreover, for any s such that $s \,\mathtt{r}\, A$, $(\lambda x\, u)s \,\mathtt{r}\, B$ we have that $\lambda x\, u \,\mathtt{r}\, A \to B$.

2. Since $u \,\mathtt{r}\, A$ and $v \,\mathtt{r}\, B$, and by **(CR1)**, we can reason by induction on the lexicographical order on the ordered pairs $(h(u), h(v))$ of the ordinal sizes of u and v. We reason then by cases on the form of the term resulting from $\langle u, v\rangle\pi_i$ for $i \in \{0, 1\}$ after one step of reduction:

   - u or v. By assumption both are reducible and hence we are done.
   - $\langle u', v\rangle\pi_i$. By **(CR2)** and since $u \,\mathtt{r}\, A$, the term u' is reducible. Since $h(u') < h(u)$, the thesis follows by induction hypothesis.
   - $\langle u, v'\rangle\pi_i$. By **(CR2)** and since $v \,\mathtt{r}\, B$, the term v' is reducible. Since $h(v') < h(v)$, the thesis follows by induction hypothesis.

   In all cases, we obtain a reducible term. By **(CR3)** we can conclude that $\langle u, v\rangle\pi_i$ for $i \in \{0, 1\}$ is reducible as well and hence $\langle u, v\rangle \,\mathtt{r}\, A \wedge B$.

□

5.5.3 The mini adequacy theorem

We prove that simply typed λ-terms that do not contain channels are reducible.

Theorem 5.5.3 (Mini Adequacy Theorem)**.** *Suppose that $w : A$ is a deterministic simply typed λ-term, with intuitionistic free variables among $x_1 : A_1, \ldots, x_n : A_n$. For all terms $t_1, \ldots, t_n$ such that*

$$\text{for } i = 1, \ldots, n, \quad t_i \,\mathtt{r}\, A_i \quad \text{we have} \quad w[t_1/x_1 \cdots t_n/x_n] \,\mathtt{r}\, A$$

Proof. This is proved exactly as Theorem 5.5.5 but without case 2., which concerns channels. □

Corollary 5.5.3.1 (Mini Strong Normalization of $\rightsquigarrow$)**.** *Suppose $t : A$ is a deterministic simply typed λ term. Then $t \,\mathtt{r}\, A$ and $t \in$ SNN.*

Proof. Assume $x_1 : A_1, \ldots, x_n : A_n$ are all the intuitionistic free variables of t and thus all its free variables. By **(CR3)** and since intuitionistic variables cannot be reduced by $\rightsquigarrow$, one has $x_i \,\mathtt{r}\, A_i$, for $i = 1, \ldots, n$. From Theorem 5.5.3, we derive that $t \,\mathtt{r}\, A$. From **(CR1)**, we conclude that $t \in$ SNN. □

By the Mini Adequacy Theorem we can prove that channels are reducible.

Proposition 5.5.4 (Reducibility of non-Deterministic Channels)**.** *For every non-deterministic channel $a : T$, $a \,\mathtt{r}\, T$.*

Proof. Since a is neutral, by **(CR3)** it is enough to show that for all u such that $a \rightsquigarrow u$, it holds that $u \,\mathtt{r}\, T$. Indeed, let us consider any u such that $a \rightsquigarrow u$; since u must be a deterministic simply typed λ-term, by Corollary 5.5.3.1 $u \,\mathtt{r}\, T$.

□

5.5.4 The adequacy theorem

We can finally prove that all simply typed λ-terms are reducible.

Theorem 5.5.5 (Adequacy Theorem)**.** *Suppose that $w : A$ is any simply typed λ-term, with intuitionistic free variables among $x_1 : A_1, \ldots, x_n : A_n$. For all terms $t_1, \ldots, t_n$ such that*

$$\text{for } i = 1, \ldots, n, \quad t_i \,\mathtt{r}\, A_i \quad \text{we have} \quad w[t_1/x_1 \cdots t_n/x_n] \,\mathtt{r}\, A$$

Proof. Notation: for any term v and formula B, we denote

$$v[t_1/x_1 \cdots t_n/x_n]$$

with $\overline{v}$. We proceed by induction on the shape of w.

1. If $w = x_i : A_i$, for some i, then $A = A_i$. So $\overline{w} = t_i \,\mathtt{r}\, A_i = A$.
2. If $w = a : A_i$, for some i and channel a, then $A = A_i$. By Proposition 5.5.4, $\overline{a} = a \,\mathtt{r}\, A$.
3. If $w = ut$, then $u : B \to A$ and $t : B$. So $\overline{w} = \overline{u}\,\overline{t} \,\mathtt{r}\, A$, for $\overline{u} \,\mathtt{r}\, B \to A$ and $\overline{t} \,\mathtt{r}\, B$ by induction hypothesis.
4. If $w = \lambda x^B u$, then $A = B \to C$. So, $\overline{w} = \lambda x^B \overline{u}$, since we may assume $x^B \neq x_1, \ldots, x_k$. For every $t \,\mathtt{r}\, B$, by induction hypothesis on u, $\overline{u}[t/x] \,\mathtt{r}\, C$. Therefore, by Proposition 5.5.2, $\lambda x^B \,\overline{u} \,\mathtt{r}\, B \to C = A$.
5. The cases of pairs and projections are straightforward.

□

Corollary 5.5.5.1 (Strong Normalization of $\rightsquigarrow$)**.** *Suppose $t : A$ is any simply typed λ-term. Then $t \in$ SNN.*

Proof. Assume $x_1 : A_1, \ldots, x_n : A_n$ are all the intuitionistic free variables of t. By **(CR3)** and since intuitionistic variables cannot be reduced by $\rightsquigarrow$, one has $x_i \,\mathtt{r}\, A_i$. From Theorem 5.5.5, we derive that $t \,\mathtt{r}\, A$. From **(CR1)**, we conclude that $t \in$ SNN. □

5.5.5 Strong normalization of $\lambda_{\parallel}$

We are finally in a position to prove the strong normalization of $\lambda_{\parallel}$.

Theorem 5.5.6 (Strong Normalization of $\lambda_{\parallel}$)**.** *For any $\lambda_{\parallel}$-term t, $t \in \mathrm{SN}_{\mathtt{p}}$.*

Proof. Assume $t = {}_a(u_1 \parallel \ldots \parallel u_m)$ and $u_1 = v_1 \parallel \ldots \parallel v_n$, …, $u_m = v_p \parallel \ldots \parallel v_q$. Define ρ_i, for $i \in \{1, \ldots, q\}$, as the ordinal size $h(v_i)$ of v_i with respect to the reduction relation $\rightsquigarrow$. We proceed by lexicographic induction on the sequence

$$\rho = (\rho_1, \ldots, \rho_q)$$

which we call the complexity of t. Let t' be any term such that $t \mapsto_{\mathtt{p}} t'$: to prove the thesis, it is enough to show that $t' \in \mathrm{SN}_{\mathtt{p}}$. Let ρ' the complexity of t'. We must consider three cases.

1. $t' = {}_a(u_1 \parallel \ldots \parallel u'_k \parallel \ldots \parallel u_m)$ and in turn

   $$u'_k = v_i \parallel \ldots \parallel v'_r \parallel \ldots \parallel v_j$$

   with $v_r \mapsto_{\mathtt{p}} v_{r'}$ by contraction of an intuitionistic redex. Then also $v_r \rightsquigarrow v'_r$. Therefore ρ' is lexicographically strictly smaller than ρ and we conclude by induction hypothesis that $t' \in \mathrm{SN}_{\mathtt{p}}$.

2. $t' = {}_a(u_1 \parallel \ldots \parallel u'_k \parallel \ldots \parallel u_m)$ and

   $$u_k = \ldots \parallel v_i \parallel \ldots \parallel v_j \parallel \ldots \qquad u'_k = \ldots \parallel v'_i \parallel \ldots \parallel v'_j \parallel \ldots$$
   $$v_i = \mathcal{D}_1[a\, w_1], \ldots v_j = \mathcal{D}_n[a\, w_n] \qquad v'_i = \mathcal{D}_1[\langle w_1, w\rangle], \ldots v_j = \mathcal{D}_n[\langle w_n, w\rangle]$$

   where w is the sequence of messages transmitted by the cross reduction. Since we have, for each l

   $$\mathcal{D}_l[a\, w_l] \rightsquigarrow \mathcal{D}_l[(\lambda x\, \langle x, w\rangle)\, w_l] \rightsquigarrow \mathcal{D}_l[\langle w_l, w\rangle]$$

   we obtain $v_i \rightsquigarrow^* v'_i, \ldots v_j \rightsquigarrow^* v'_j$. Therefore ρ' is lexicographically strictly smaller than ρ and we conclude by induction hypothesis that $t' \in \mathrm{SN}_{\mathtt{p}}$.

3. $t' = v_{i_1} \parallel \ldots \parallel v_{i_k}$. As $v_{i_1}, \ldots, v_{i_k}$ all belong to $\mathrm{SN}_{\mathtt{p}}$, we easily obtain $t' \in \mathrm{SN}_{\mathtt{p}}$.

□

5.6 Computing with $\lambda_{\parallel}$

We illustrate the expressive power of $\lambda_{\parallel}$ with examples of programs. The first example shows that we can formulate in $\lambda_{\parallel}$ a term which behaves similarly to parallel OR, and thus that $\lambda_{\parallel}$ is more powerful than simply typed λ-calculus. The next two are parallel programming examples featuring a numeric approximation of π and an algorithm for computing shortest paths on graphs.

All $\lambda_{\|}$-terms that we are going to construct will be reduced according to a fixed normalization strategy, which we are going to define now: a recipe for selecting, in any given term, the subterm to which apply one of our reductions. By strong normalization of $\lambda_{\|}$, we already know it will terminate.

Our normalization strategy follows the following rules, whose goal is to make parallel programming with $\lambda_{\|}$ as efficient and as deterministic as possible.

(a) Messages should be normalized before being sent. This property is fundamental for efficient parallel computation. Indeed, for instance, a repeatedly forwarded message can be duplicated many times inside a process network, because each process may keep a copy of it; hence if the message is not normal, each process may have to normalize it all over again, wasting resources.

(b) The senders and the receiver of the message should be normalized before the communication. Indeed, the communication behavior depends on the syntactic shape of a process: it is the rightmost occurrence of the channel in a $\lambda_{\|}$-term to determine what message is sent or received. Hence, a term might send a message while its normal form another one. For example $(\lambda x\, y\, x\, (an))(am)$ would transmit m, while its normal form $y\, (am)\, (an)$ would transmit n. We want to avoid this kind of unpredictable behavior and make clear what channel is supposed to be the active one, which sends or receives a message.

(c) The reduction strategy should exploit parallelism as much as possible.

Informally, our normalization strategy consists in iterating the basic reduction relation $\succ_{\mathtt{p}}$ defined below, that takes a term $t = {}_a(u_1 \parallel \ldots \parallel u_m)$ and performs the following operations:

1) We select all the threads of $u_1, \ldots, u_m$ that will send or receive the next message and we let them complete their internal computations. This way, we satisfy (a) and (b). We also exploit parallelism as much as possible, and thus (c).

2) We let the selected threads transmit their message.

3) While executing 1) and 2); to avoid inactivity, we let the other threads perform some other independent calculations that may be carried out in parallel. Hence we satisfy (c).

4) If the previous operations are not possible, we extract the results, if any.

Definition 5.6.1 (Reduction Strategy $\succ_{\mathtt{p}}$)**.** Let $t = {}_a(u_1 \parallel \ldots \parallel u_m)$ be a $\lambda_{\|}$-term. We write $t \succ_{\mathtt{p}} t'$ whenever t' has been obtained from t by applying one of the following:

1. We select a receiver u_i for the next communication. We normalize, among the threads that contain a, those that are rightmost in u_i or in a process outlinked to u_i. If now it is possible, we apply a mini cross reduction

$$ {}_a(u_1 \parallel \ldots \parallel u_i \parallel \ldots \parallel u_m) \mapsto_{\mathtt{p}} {}_a(u_1 \parallel \ldots \parallel u_i' \parallel \ldots \parallel u_m) $$

   followed by some intuitionistic reductions.

2. Provided that by 1. it is impossible to have a reduction $t \mapsto_{\mathtt{p}}^{*} t'$ where $t' \neq t$, we apply, if possible, a simplification reduction. We then normalize the remaining simply typed λ-terms.

We can reduce every $\lambda_{\|}$-term in normal form just by iterating the reduction relation $\succ_{\mathtt{p}}$. Indeed, if any communication is possible, by 1. we can select a suitable receiver and apply a mini cross reduction. If no communication or intuitionistic reduction is possible but a simplification is possible, 2. applies and we can simplify the term. Otherwise, we can just normalize the simply typed λ-terms by 1. or 2.

This normalization strategy leaves some room for non-determinism: it prescribes when a communication reduction should be fired, but does not select a process out of those that can potentially receive messages, thus leaving a number of possible ways of actually performing the communication. To limit this non-determinism, we underline exactly one process and impose that only an underlined process $\underline{\mathcal{C}[a^{F_i \to F_i \wedge G_j}\, t]}$ can receive a message. Immediately after the reception of the message, or if no message can be received, we underline either the next process on the right, if any, or the first one from left otherwise. Another source of non-determinism is due to simplification reductions

$$\overset{\mathcal{A}}{a}((u_{m_1} \| \ldots \| u_{n_1}) \| \ldots \| (u_{m_p} \| \ldots \| u_{n_p})) \mapsto_{\mathtt{p}} u_{j_1} \| \ldots \| u_{j_q}$$

In this case, we impose that $u_{j_1} \| \ldots \| u_{j_q}$ results by selecting from each $(u_{m_i} \| \ldots \| u_{n_i})$, with $1 \leq i \leq p$, the leftmost thread not containing a.

5.6.1 Parallel OR

Berry's sequentiality theorem [Bar84] implies that there is no parallel OR, namely a simply typed λ-term $O : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ such that $O\,\mathtt{ff}\,\mathtt{ff} \mapsto_{\mathtt{p}}^{*} \mathtt{ff}$, $O\,u\,\mathtt{tt} \mapsto_{\mathtt{p}}^{*} \mathtt{tt}$, $O\,\mathtt{tt}\,u \mapsto_{\mathtt{p}}^{*} \mathtt{tt}$ for *every* λ-term u, where $\mathtt{tt}, \mathtt{ff}$ are the boolean constants. As a consequence, there cannot be a λ-term O such that $\mathsf{O}[\mathtt{ff}/x][\mathtt{ff}/y] \mapsto_{\mathtt{p}}^{*} \mathtt{ff}$, $\mathsf{O}[u/x][\mathtt{tt}/y] \mapsto_{\mathtt{p}}^{*} \mathtt{tt}$, $\mathsf{O}[\mathtt{tt}/x][u/y] \mapsto_{\mathtt{p}}^{*} \mathtt{tt}$ for every λ-term u. A term with the above property can instead be defined in $\lambda_{\|}$. To implement this term we need to process the two inputs in parallel. If both inputs evaluate to $\mathtt{ff}$, though, at least one process needs to have all the information in order to output the result $\mathtt{ff}$. Hence a simple channel of the form

is enough. The procedure in Table 5.3 extracts from this graph the axiom $\mathrm{EM}' = (A \to A \wedge \bot) \vee (B \to B \wedge A)$. We recall that the corresponding mini cross reduction is

$$\overset{}{a}(\mathcal{C}[aw] \| (\mathcal{D}_1[av_1] \| \ldots \| \mathcal{D}_n[av_n])) \mapsto_{\mathtt{p}} \overset{}{a}(\mathcal{C}[aw] \| (\mathcal{D}_1[\langle v_1, w\rangle] \| \ldots \| \mathcal{D}_n[\langle v_n, w\rangle]))$$

We add to $\lambda_{\|}$ the boolean type, $\mathtt{tt}, \mathtt{ff}$ and the usual "$\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$" construct [GLT89]. We define in $\lambda_{\|}$

$$\mathsf{O} := \overset{}{a}(\mathsf{if}\ x\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ a\,\mathtt{ff}\,\pi_0 \| \underline{\mathsf{if}\ y\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ a * \pi_1})$$

where we assume that $a : \mathsf{Bool} \to \mathsf{Bool} \wedge \bot$ in the first process, that $a : \top \to \top \wedge \mathsf{Bool}$ in the second one, and that $* : \top \ = \ \lambda x^{\bot}\, x : \top$. Now, on one hand

$$\begin{aligned}\mathsf{O}[u/x][\mathtt{tt}/y] =_a (\mathsf{if}\ u\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ a\,\mathtt{ff}\pi_0 \parallel \underline{\mathsf{if}\ \mathtt{tt}\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ a\ *\ \pi_1}) \\ \mapsto^*_{\mathtt{p}} {}_a(\mathsf{if}\ u\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ a\,\mathtt{ff}\pi_0 \parallel \underline{\mathtt{tt}}) \ \mapsto_{\mathtt{p}}\ \mathtt{tt}\end{aligned}$$

and symmetrically $\mathsf{O}[\mathtt{tt}/x][u/y] \mapsto^*_{\mathtt{p}} \mathtt{tt}$. On the other hand,

$$\mathsf{O}[\mathtt{ff}/x][\mathtt{ff}/y] \mapsto^*_{\mathtt{p}} {}_a(\mathsf{if}\ \mathtt{ff}\ \mathsf{then}\ \mathtt{ff}\ \mathsf{else}\ a\,\mathtt{ff}\ \pi_0 \parallel \underline{\mathsf{if}\ \mathtt{ff}\ \mathsf{then}\ \mathtt{ff}\ \mathsf{else}\ a\ *\ \pi_1}) \mapsto^*_{\mathtt{p}}$$

$${}_a(a\,\mathtt{ff}\ \pi_0 \parallel \underline{a\ *\ \pi_1}) \mapsto_{\mathtt{p}}\ {}_a(a\,\mathtt{ff}\ \pi_0 \parallel \langle *, \mathtt{ff}\rangle\, \pi_1) \ \mapsto_{\mathtt{p}}\ \langle *, \mathtt{ff}\rangle\, \pi_1 \mapsto_{\mathtt{p}} \mathtt{ff}$$

5.6.2 A parallel program for computing π

We implement in $\lambda_{\parallel}$ a parallel program for computing an arbitrarily precise approximation of π. As is well know π can be computed as the limit of the following summation

$$\pi = \lim_{l\to\infty} \frac{1}{l} \sum_{i=1}^{l} f(\frac{i-\frac{1}{2}}{l}) \quad \text{where} \quad f(x) = \frac{4}{1+x^2}$$

For any given l, instead of calculating sequentially the whole sum $\sum_{i=1}^{l} f(\frac{i-1/2}{l})$ and then dividing it by l, it is more efficient to distribute different parts of the sum to p parallel processes. When the processes terminate, they can send the results $m_1, \ldots, m_p$, as shown below, to a process which computes the final result $\frac{1}{l}(m_1 + \ldots + m_p)$, see also [Loo11].

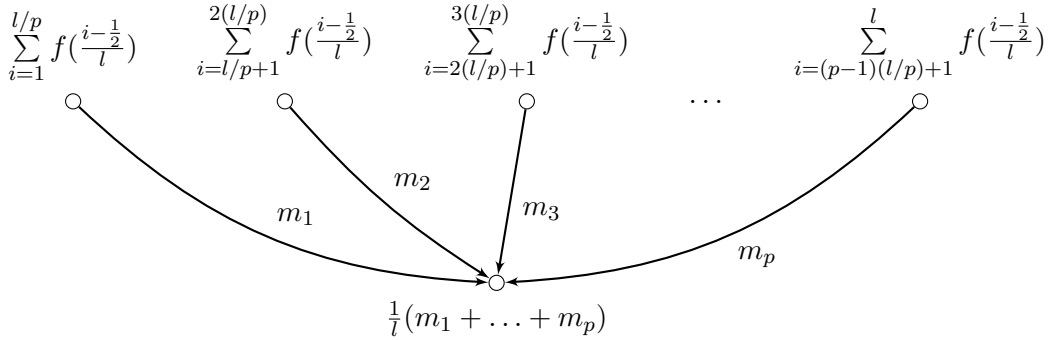


Figure 5.2

The graph in Figure 5.2, in which we omit reflexive edges, is encoded by the axiom $\mathcal{A} = (A_1 \to A_1 \wedge \bot) \vee \ldots \vee (A_p \to A_p \wedge \bot) \vee B \to (B \wedge A_1 \wedge \ldots \wedge A_p)$ (cf. the procedure in Table 5.3). To write the program, we add to $\lambda_{\parallel}$ types and constants for rational numbers, together with function constants

$$\mathsf{f}_k\, l \mapsto^*_{\mathtt{p}} \sum_{i=(k-1)l/p+1}^{(kl/p)} f(\frac{i-\frac{1}{2}}{l}) \quad \text{and} \quad \mathsf{sum}\,\langle n_1, \ldots, n_i\rangle\, l \mapsto^*_{\mathtt{p}} \frac{1}{l}(n_1 + \ldots + n_i)$$

computing respectively the p partial sums and the final result. The $\lambda_{\|}$-term that takes as input the length of the summation to be carried out and yields the corresponding approximation of π is:

$$ {}_a(\ a(\mathsf{f}_1\, l)\pi_0 \parallel \ldots \parallel a(\mathsf{f}_p\, l)\pi_0 \parallel \mathsf{sum}\,(a\,(\lambda x^{\perp}\, x)\,\pi_1)\, l\ )$$

By instantiating $\mathbb{A}_1, \ldots, \mathbb{A}_p$ in the extracted $\mathcal{A}$ with the type Q of rational numbers, we type all displayed occurrences of a in the first p threads by $\mathsf{Q} \to \mathsf{Q} \wedge \perp$, and the last by $\top \to (\top \wedge \mathsf{Q} \wedge \ldots \wedge \mathsf{Q})$. Given any multiple n of p, we have

$$ ({}_a(\ a(\mathsf{f}_1\, l)\,\pi_0 \parallel \ldots \parallel a(\mathsf{f}_p\, l)\,\pi_0 \parallel \underline{\mathsf{sum}\,(a\,(\lambda x^{\perp}\, x)\,\pi_1)\, l}\ ))[n/l]$$

$$ =\ {}_a(\ a(\mathsf{f}_1\, n)\,\pi_0 \parallel \ldots \parallel a(\mathsf{f}_p\, n)\,\pi_0 \parallel \underline{\mathsf{sum}\,(a\,(\lambda x^{\perp}\, x)\,\pi_1)\, n}\ )$$

$$ \mapsto_{\mathsf{p}}\ {}_a(\ a(\sum_{i=1}^{(n/p)} f(\frac{i-\frac{1}{2}}{l}))\,\pi_0 \parallel \ldots \parallel a(\sum_{i=(p-1)n/p+1}^{n} f(\frac{i-\frac{1}{2}}{l}))\,\pi_0 \parallel \underline{\mathsf{sum}\,(a\,(\lambda x^{\perp}\, x)\,\pi_1)\, n}\ )$$

$$ \mapsto^{*}_{\mathsf{p}}\ \mathsf{sum}\,(\langle \sum_{i=1}^{(n/p)} f(\frac{i-\frac{1}{2}}{l}))), \ldots, \sum_{i=(p-1)n/p+1}^{n} f(\frac{i-\frac{1}{2}}{l}))\rangle\, n \qquad \mapsto^{*}_{\mathsf{p}}\ \frac{1}{n}\sum_{i=1}^{n} f(\frac{i-\frac{1}{2}}{n})$$

Notice that a single $\lambda_{\|}$-term cannot iterate the procedure for increasing values of l. Indeed, each $\lambda_{\|}$-term has a fixed communication topology.

The $\lambda_{\|}$-terms in the next two examples implement the corresponding algorithms in [Loo11].

5.6.3 A parallel Floyd–Warshall algorithm

We define a $\lambda_{\|}$-term that implements a parallel version of the Floyd–Warshall algorithm, a well known procedure for computing shortest paths in graphs. The algorithm takes as input an oriented graph and outputs a matrix containing the length of the shortest path between each pair of nodes.

Formally, the input graph is coded as a matrix $I(0)$ and the nodes of the graphs are labeled as $1, \ldots, n$. Then the sequential Floyd–Warshall algorithm computes a sequence of matrixes $I(1), \ldots, I(n)$, representing closer and closer approximations of the desired output matrix. In particular, the entry (i, j) of $I(k)$ is the length of the shortest path connecting i and j such that every node of the path, except for the endpoints, is among the nodes $1, 2, \ldots k$. Now, each matrix $I(k)$ can be easily computed from $I(k-1)$. The idea is that passing through the node k might be better than passing only through the first $k-1$ nodes, or not. Therefore in order to compute $I(k+1)$ from $I(k)$, one only needs to evaluate the right-hand side of the following equation:

$$ I_{i,j}(k) = \mathsf{min}(I_{i,j}(k-1), I_{i,k}(k-1) + I_{k,j}(k-1)) \tag{5.2}$$

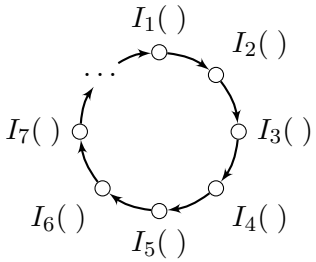
Indeed, $I_{i,j}(k-1)$ represents the shortest path between i, j passing only through the first $k-1$ nodes, while $I_{i,k}(k-1) + I_{k,j}(k-1)$ computes the shortest path between i, j that passes through k and the first $k-1$ nodes.

To speed-up this computation, we can execute it in parallel. For each row of the matrix, we create a different parallel process. Each parallel process will compute the corresponding row. Let us say that the i-th process has to compute the i-th row $I_i(k)$. What information does ith-process need? Actually, only two rows: $I_i(k-1)$ and $I_k(k-1)$. Therefore at each round k the process-i only lacks the row $I_k(k-1)$ to perform its computation. This row can be communicated to process-i by the process-k, which is in charge to compute that row. These considerations lead to a well-known parallel version of the Floyd–Warshall algorithm.

The ring Floyd–Warshall algorithm

1. Take the input $n \times n$-matrix $I(0)$ and distribute the i-th row $I_i(0)$ to process-i. Organize the n processes in a ring structure such as the following, see [Loo11], omitting reflexive edges:

(∗)



2. For $k = 1$ to n, starting from process-k, let all the processes forward the row $I_k(k-1)$ to their successors in the ring, until the process $k+1$ receives again the same row. After the row has circulated, let the processes compute in parallel the rows of the matrix $I(k)$.

3. Let $I(n)$ be the output.

The idea of the ring Floyd–Warshall Algorithm is that, at any stage k, communicating the required row $I_k(k-1)$ through the ring structure requires comparatively little time compared to computing each row of the matrix, therefore the overhead of the communication is compensated by the speed-up in the matrix computation.

A $\lambda_{\|}$ program for the Floyd–Warshall algorithm

In order to write a $\lambda_{\|}$ program that computes the Ring Floyd–Warshall Algorithm, we add integers to $\lambda_{\|}$, as usual, and we denote with A the function type corresponding to the rows of the matrixes $I(k)$ computed by the algorithm. The expression $I_x(y)$ represents the function computing the x^{th} line of the matrix at the y^{th} stage of the algorithm. We assume that the value of $I_x(y)$ also contains information about the line number x and the stage number y. Finally, we add the constant $f : A \wedge A \wedge S \to A$ for the function such that:

- with input $\langle I_i(k-1), I_k(k-1), \delta\rangle$ outputs $\langle I_i(k), \delta\rangle$
- for any other pair $\langle I_z(l), I_m(n), \delta\rangle$, outputs $\langle I_z(l), \delta\rangle$

The first line implements the calculations needed for equation (5.2). The second line comes into play at the end of each iteration of the algorithm, when the process that receives twice the same row just discards it and starts sending its own row, thus beginning the next iteration of the algorithm. The third argument $\delta : S$ of f is a dummy term such that S is not a subformula of A and it is just introduced to activate the communication.

As handful notation, for any three terms u, v, s we define $(u, v)^1 s$ as $u(vs)$, and $(u, v)^{n+1} s$ for $n > 0$ as $u(v((u, v)^n s))$. Moreover, for any two terms u, s we define $(u, \pi_i)^1 s$ as $u(s\,\pi_i)$, and $(u, \pi_i)^{n+1} s$ for $n > 0$ as $u(((u, \pi_i)^n s)\pi_i)$. Intuitively, the notation $(u, v)^n s$ represents what we obtain if we take a term s and then apply alternately v and u to the term resulting from the last operation. We stop only when we have n applications of u and n applications of v. We obtain, ultimately, a term of the form $u(v(u(v(\dots u(vs)\dots))))$ in which and u and v occur n times each. The notation $(u, \pi_i)^n s$ is analogous, but instead of applying v we apply the projection π_i.

We are now ready to define the n processes that are run in parallel during the execution of the ring Floyd–Warshall algorithm:
Process p_1:

$$((f, a)^n \langle I_1(0), \delta\rangle)\pi_0 \parallel ((a, \pi_1)^{n+1} \langle 0, I_1(0), \delta\rangle)\pi_0\pi_0 \parallel ((a, \pi_1)^1 \langle 0, \langle I_1(0), \delta\rangle\rangle)\pi_0\pi_0$$

Process p_i, with $1 < i < n$:

$$\begin{aligned}&((f, a)^{n+1} \langle I_i(0), \delta\rangle)\pi_0 \parallel ((a, \pi_1)^{n+2} \langle 0, \langle I_i(0), \delta\rangle\rangle)\pi_0\pi_0\\ &\quad \parallel (a\,((f, a)^i \langle I_i(0), \delta\rangle))\pi_0\pi_0 \parallel ((a, \pi_1)^i \langle 0, \langle I_i(0), \delta\rangle\rangle)\pi_0\pi_0\end{aligned}$$

Process p_n:

$$\begin{aligned}&((f, a)^{n+1} \langle I_i(0), \delta\rangle)\pi_0 \parallel ((a, \pi_1)^{n+1} \langle 0, \langle I_i(0), \delta\rangle\rangle)\pi_0\pi_0\\ &\quad \parallel (a\,((f, a)^i \langle I_i(0), \delta\rangle))\pi_0\pi_0 \parallel ((a, \pi_1)^i \langle 0, \langle I_i(0), \delta\rangle\rangle)\pi_0\pi_0\end{aligned}$$

For an intuitive reading of the parts of the terms above, consider the notation $(f, a)^m\, t$. This notation represents a term of the form $f(a(\dots f(at)\dots))$. Only one operation can be immediately performed with a term like this: using the innermost application of a. Thus we can either transmit t through a and then consume the application at to receive a message, or we can just consume at to receive a message. The received message will be the argument of the innermost f. The value that f computes, in turn, will be the argument of the next communication channel. Therefore, terms of this form alternate two phases: one in which they can send and receive, and one in which they apply f to the received message. Most of the times the rightmost communication channel of a process p_i is not contained in the terms of the form $(f, a)^m\, t$. In this case, the terms $(f, a)^m\, t$ use their innermost channel application only to receive a message. Afterwards, as usual, they apply f to the received message. The terms $(a, \pi_1)^m\, t$ have the form $a((\dots a(t\pi_1)\dots)\pi_1)$. These terms project, send and receive; and then start over. The projections are used to select only certain messages from the tuple of received messages. The selected messages are not used, but just forwarded to another process.

The global intuition is that each process p_i, with $i > 1$, is a parallel composition of four threads: the first and the third have the task of computing the rows $I_i(k)$, while the second and the fourth have the task of sending and forwarding rows. The process p_1 behaves in the same way, but has three threads: the first computes the rows $I_1(k)$, the second receives and forwards rows, while the thirds sends its own row.

The term implementing the ring Floyd–Warshall algorithm is defined as

$$_a(\, p_1 \parallel \ldots \parallel p_n)$$

The axiom extracted from the ring structure $(*)$ above by the procedure in Table 5.3 is

$$\mathcal{A} = (A_1 \to A_1 \wedge A_n) \vee (A_2 \to A_2 \wedge A_1) \vee \ldots \vee (A_{n-1} \to A_{n-1} \wedge A_{n-2}) \vee (A_n \to A_n \wedge A_{n-1})$$

We can thus let $A_1 = \ldots = A_n = (A \wedge S)$ in $\mathcal{A}$ and type all displayed occurrences of a by $(A \wedge S) \to (A \wedge S) \wedge (A \wedge S)$.

Let us consider an example of execution. As the dummy terms δ only influence the activation of the communications without changing the rest of the computation – and especially without changing the result – for the sake of simplicity we omit both them and the associated projections. In the general case, the terms are simplified into
Process p'_1:

$$(f,a)^n\, I_1(0) \parallel ((a,\pi_1)^{n+1}\, \langle 0, I_1(0)\rangle)\pi_0 \parallel ((a,\pi_1)^1\, \langle 0, I_1(0)\rangle)\pi_0$$

Process p'_i, with $1 < i < n$:

$$(f,a)^{n+1}\, I_i(0) \parallel ((a,\pi_1)^{n+2}\, \langle 0, I_i(0)\rangle)\pi_0 \parallel (a\,(f,a)^i\, I_i(0))\pi_0 \parallel ((a,\pi_1)^i\, \langle 0, I_i(0)\rangle)\pi_0$$

Process p'_n:

$$(f,a)^{n+1}\, I_i(0) \parallel ((a,\pi_1)^{n+1}\, \langle 0, I_i(0)\rangle)\pi_0 \parallel (a\,((f,a)^i\, I_i(0)))\pi_0 \parallel ((a,\pi_1)^i\, \langle 0, I_i(0)\rangle)\pi_0$$

The simplified three-process instance $_a(\, p'_1 \parallel p'_2 \parallel p'_3)$ is

$$\begin{aligned}
&{}_a\big(\big(f(a(f(a(f(aI_1(0)))))) \parallel (a((a((a((a(\langle 0, I_1(0)\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_0 \\
&\quad \parallel (a(\langle 0, I_1(0)\rangle\pi_1))\pi_0\big) \\
&\parallel \underline{\big(f(a(f(a(f(a(f(aI_2(0))))))))) \parallel (a((a((a((a((a(\langle 0, I_2(0)\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_1))\pi_0} \\
&\quad \underline{\parallel (a(f(a(f(aI_2(0)))))))\pi_0 \parallel (a((a(\langle 0, I_2(0)\rangle\pi_1))\pi_1))\pi_0\big)} \\
&\parallel \big(f(a(f(a(f(a(f(aI_3(0))))))))) \parallel (a((a((a((a(\langle 0, I_3(0)\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_0 \\
&\quad \parallel (a(f(a(f(a(f(aI_3(0))))))))\pi_0 \parallel (a((a((a(\langle 0, I_3(0)\rangle\pi_1))\pi_1))\pi_1))\pi_0\big)\big)
\end{aligned}$$

According to our normalization strategy we will normalize the threads sending messages right before they communicate. Afterwards, we normalize the rightmost threads receiving the messages. All other intuitionistic reductions are performed in-between communications. Finally, we make sure to contract all intuitionistic redexes before the beginning of a new iteration of the algorithm.

We start underlining the second process to mean that it is under focus and hence will receive the first communication:

$$\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\big(\big(f(a(f(a(f(a(f(aI_1(0)))))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (aI_1(0))\pi_0\big)\\
&\quad \parallel \underline{\big(f(a(f(a(f(a(f(a(f(aI_2(0))))))))) \parallel (a((a((a((a((aI_2(0))\pi_1))\pi_1))\pi_1))\pi_1))\pi_0}\\
&\quad\quad \underline{\parallel (a(f(a(f(aI_2(0)))))))\pi_0 \parallel (a((aI_2(0))\pi_1))\pi_0\big)}\\
&\quad \parallel \big(f(a(f(a(f(a(f(a(f(aI_3(0))))))))) \parallel (a((a((a((a\langle 0, I_3(0)\rangle)\pi_1))\pi_1))\pi_1))\pi_0\\
&\quad\quad \parallel (a(f(a(f(a(f(a(f(aI_3(0))))))))\pi_0 \parallel (a((a((a\langle 0, I_3(0)\rangle)\pi_1))\pi_1))\pi_0\big)\big)
\end{aligned}$$

We then transmit the value $I_1(0)$, (we display its relevant occurrences between * *) from the first process to the second process and we move the focus to the next term:

$$\begin{aligned}
&{}_{a}\big(\big(f(a(f(a(f(aI_1(0)))))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel ({}^\star aI_1(0)^\star)\pi_0\big)\\
&\parallel \big(f(a(f(a(f(a(f\langle I_2(0), {}^\star I_1(0)^\star\rangle)))))))) \parallel (a((a((a((a((\langle I_2(0), {}^\star I_1(0)^\star\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_1))\pi_0\\
&\quad \parallel (a(f(a(f\langle I_2(0), {}^\star I_1(0)^\star\rangle)))))\pi_0 \parallel (a(\langle I_2(0), {}^\star I_1(0)^\star\rangle\pi_1))\pi_0\big)\\
&\parallel \underline{\big(f(a(f(a(f(a(f(aI_3(0))))))))) \parallel (a((a((a((a\langle 0, I_3(0)\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_0}\\
&\quad \parallel \underline{(a(f(a(f(a(f(aI_3(0))))))))\pi_0 \parallel (a((a((a\langle 0, I_3(0)\rangle\pi_1))\pi_1))\pi_1))\pi_0}\big)\big)
\end{aligned}$$

Notice that $I_2(0)$ is not destroyed by the communication but saved using the memorization mechanism. This term is needed, indeed, by the function f in order to compute $I_2(1)$. We normalize the receiver of the previous communication and keep normalizing the other redexes in parallel, thus the rightmost thread of the second process become ready to forward the value $I_1(0)$ to the third thread:

$$\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\big(\big(f(a(f(a(f(aI_1(0)))))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(I_1(0)))\big)\pi_0\\
&\quad \parallel \big(f(a(f(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0\\
&\quad\quad \parallel (a(f(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))))\pi_0 \parallel ({}^\star aI_1(0)^\star)\pi_0\big)\\
&\quad \parallel \underline{\big(f(a(f(a(f(a(f(aI_3(0))))))))) \parallel (a((a((a((aI_3(0))\pi_1))\pi_1))\pi_1))\pi_0}\\
&\quad\quad \parallel \underline{(a(f(a(f(a(f(aI_3(0))))))))\pi_0 \parallel (a((a((aI_3(0))\pi_1))\pi_1))\pi_0}\big)\big)
\end{aligned}$$

The third thread receives $I_1(0)$ and the procedure continues the computation:

$$\mapsto_{\mathtt{p}} \cdots$$

At the end of the first of the three cycles, the second process, under focus, receives $I_1(0)$ again:

$$\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\big(\big(f(a(f(aI_1(1))))) \parallel (a((a(({}^\star aI_1(0)^\star)\pi_1))\pi_1))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)\\
&\quad \parallel \big(f(a(f(a(f\langle I_2(1), {}^\star I_1(0)^\star\rangle)))))) \parallel (a((a((a(\langle I_1(0), {}^\star I_1(0)^\star\rangle\pi_1)\pi_1))\pi_1))\pi_0\\
&\quad\quad \parallel (a(f\langle I_2(1), {}^\star I_1(0)^\star\rangle))))\pi_0 \parallel \langle I_1(0), {}^\star I_1(0)^\star\rangle\pi_0\big)\\
&\quad \parallel \big(\underline{f(a(f(a(f(aI_3(1))))))) \parallel (a((a((a(I_1(0)))\pi_1))\pi_1))\pi_0 \parallel (a(f(a(f(aI_3(1))))))\pi_0}\\
&\quad\quad \underline{\parallel (a((aI_1(0))\pi_1))\pi_0}\big)\big)
\end{aligned}$$

The reception of $I_1(0)$ exhausts all forwarding channels of the second process – those in the rightmost thread – and triggers the communication of the value $I_2(1)$ just computed,

thus starting the second cycle:

$$\begin{aligned}
&\mapsto_{\mathrm{p}\,a}^{*}\big(\big(f(a(f(aI_1(1)))) \parallel (a((a((aI_1(0))\pi_1))\pi_1))\pi_0 \parallel (\langle I_1(0), I_1(0)\rangle)\pi_0\big)\\
&\quad \parallel \big(f(a(f(aI_2(1)))) \parallel (a((a((aI_1(0))\pi_1))\pi_1))\pi_0 \parallel ({}^{\star}aI_2(1)^{\star})\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)\\
&\quad \parallel \big(\underline{f(a(f(a(f(aI_3(1))))))) \parallel (a((a((a(I_1(0)))\pi_1))\pi_1))\pi_0}\\
&\qquad \underline{\parallel (a(f(a(f(aI_3(1))))))\pi_0 \parallel (a((aI_1(0))\pi_1))\pi_0}\big)\big)\\
&\mapsto_{\mathrm{p}} \dots
\end{aligned}$$

The reduction continues in the same fashion for two other cycles, until the third process does not contain any occurrence of a anymore:

$$\begin{aligned}
&\mapsto_{\mathrm{p}\,a}^{*}\big(\big(\underline{f\langle I_1(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)\\
&\quad \parallel \big(f\langle I_2(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)\\
&\quad \parallel \big(f\langle I_3(2), I_3(2)\rangle \parallel \langle I_2(1), I_3(2)\rangle\pi_0 \parallel \langle I_3(2), I_3(2)\rangle\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)
\end{aligned}$$

We then compute all occurrences of f in parallel:

$$\begin{aligned}
&\mapsto_{\mathrm{p}\,a}^{*}\big(\big(\underline{I_1(3) \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)\\
&\quad \parallel \big(I_2(3) \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)\\
&\quad \parallel \big(I_3(3) \parallel \langle I_2(1), I_3(2)\rangle\pi_0 \parallel \langle I_3(2), I_3(2)\rangle\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)
\end{aligned}$$

and keep only the leftmost threads

$$\mapsto_{\mathrm{p}}^{*} I_1(3) \parallel I_2(3) \parallel I_3(3)$$

See the appendix for the unabridged version of this reduction.

5.7 Related work

As in the case of λ_{Cl} and λ_{G}, a comparison with π-calculus is possible, and due, for $\lambda_{\parallel}$ as well. The main similarities and differences between π-calculus and the calculi λ_{Cl} and λ_{G} can be mentioned also with respect to $\lambda_{\parallel}$. In particular, the binder ${}_a(\ )$ in ${}_a((u_1 \parallel \dots \parallel u_n) \parallel \dots \parallel (u_p \parallel \dots \parallel u_q))$ and νa in a π-calculus term $\nu a(P \mid Q)$ have the same rôle. On the other hand, while in π-calculus communication only applies to names, communication in $\lambda_{\parallel}$ is higher-order. Moreover, π-calculus is meant to model concurrent systems and $\lambda_{\parallel}$ is meant to encode parallel algorithms. Hence, concurrency in $\lambda_{\parallel}$ is much more limited than in π-calculus.

A formalism similar to $\lambda_{\parallel}$ is presented in [Bou89]. This system is a concurrent extension of λ-calculus that features communication mechanisms inspired by process calculi. The obvious difference between [Bou89] and the present work is the use of logic, which plays no rôle in the former but is essential for the latter.

As for typed concurrent extensions of λ-calculus, the type systems of most of them are based on linear logic. We cite as prominent examples the functional languages presented in [Wad12] and [TCP13]. Both of them are based on Curry–Howard correspondences between proof systems for linear logic and extensions of λ-calculus by primitives for π-calculus-style communications [CP10b]. In particular, the work in [Wad12] presents a

translation from a functional language extended by primitive operators for communication into a process calculus with session types which corresponds to a sequent calculus for classical linear logic. The system in [TCP13] is based on the interpretation presented in [CP10b] of a sequent calculus for dual intuitionistic linear logic into π-calculus with session types. This computational interpretation is then integrated in a functional language. Both the type system and the programming language presented in in [TCP13] have distinct functional and concurrent parts. Our calculi, by contrast, exploit natural deduction systems for intermediate logics to type the terms of a parallel λ-calculus. Unlike session types, the types based on intermediate logics that are used in our calculi do not represent dynamic aspects of the communications.

A recent work [CMS18] presents a Curry–Howard correspondence between a fragment of linear logic and a process-calculus-style formalism combining π-calculus and annotations for communication choreographies. The proof system used as type system for this language extends linear logic sequents with two structural connectives. One of them composes sequents in parallel similarly to the | operator used in hypersequents; the other one is used to encode relations between parallel sequents. The object constructed using these structural connectives on sequents represents the topology of a network. The deduction rules acting across different sequents correspond to the operators encoding choreography annotations. The types used in $\lambda_{\|}$ do not contain information about the dynamical aspects of choreographies. Using these types we can only express static properties of the communication topologies. It is not know, however, whether $\lambda_{\|}$ can be simulated by the calculus presented in [CMS18].

Finally, a conservative extension of classical linear logic is defined in [KMP19] by introducing a hypersequent-style connective | in a single sided sequent calculus. A proofs-as-programs correspondence with π-calculus is then shown. While both in [KMP19] and in the present work the structural connective | is computationally interpreted – directly or indirectly – as a parallelism operator, the two works differ with respect to the logical interpretation of this connective. In [KMP19], it corresponds to the linear logic multiplicative conjunction $\otimes$, in the hypersequent calculi on which our work is based, the structural connective | corresponds to disjunction.

CHAPTER 6

A computational interpretation of intermediate logics

We define the framework $\lambda_{\|\mathrm{L}}$ of logic-based concurrent λ-calculi. The aim of $\lambda_{\|\mathrm{L}}$ is to provide a computational interpretation of the intermediate logics that can be defined extending intuitionistic logic by disjunctive tautologies of the form

$$\mathcal{A}_{\mathtt{i}} = (F_1 \to G_1) \vee \ldots \vee (F_m \to G_m)$$

where for every $F_i \neq \top$, $F_i = G_j$ for some j and F_i is not repeated – the subscript L in $\lambda_{\|\mathrm{L}}$ precisely ranges over these intermediate logics.

While the calculus $\lambda_{\|}$ presented in Chapter 5 is very well suited for encoding parallel algorithms; technically, it does not fully correspond to the intermediate logics that we can axiomatize by the considered disjunctive tautologies. Indeed, the restrictions on the number of parallel operators binding communication variables and the absence of general cross reductions constitute a proper limitation of the logical systems interpreted by the calculus. Moreover, in many of the considered logics disjunction is not a definable connective. Hence, we cannot base on $\mathrm{N}\mathfrak{I}^{\to}_{\wedge\bot}$ a computational interpretation of these logics, but we need to explicitly add disjunction rules and thus base the relative λ-calculi on the Curry–Howard correspondence for $\mathrm{N}\mathfrak{I}$.

The $\lambda_{\|\mathrm{L}}$ framework, as opposed to $\lambda_{\|}$, enables us to define Curry–Howard correspondences between typed concurrent λ-calculi and complete natural deduction calculi for the considered intermediate logics. Furthermore, for normal $\lambda_{\|\mathrm{L}}$ proof terms we are able to show that the subformula property holds. Hence, by providing a general normalization result for $\lambda_{\|\mathrm{L}}$ we finally confirm, for a rather general setting, Avron's 1991 thesis on the relationship between concurrent computation and the intermediate logics that are naturally formalized as hypersequents calculi – see Section 2.4 for a thorough discussion of this thesis.

In presenting $\lambda_{\|\mathrm{L}}$ we also introduce a set of reduction rules whose conditions are solely based on the shape of terms and do not rely on their types. The definition of such a reduction system might also shed light on the long-standing problem of defining a well-behaved untyped concurrent λ-calculus.

The content of the present chapter is based on [ACG19a]. In Section 6.1 we present the type systems $\lambda_{\|\mathrm{L}}$ and in Section 6.1.1 we present their reduction rules. We prove a general normalization result in Section 6.2 and, finally, in Section 6.3 we prove that any normal proof term of $\lambda_{\|\mathrm{L}}$ enjoys the subformula property.

6.1 The type system of $\lambda_{\|\mathrm{L}}$ and its reduction rules

We introduce modular natural deduction calculi extending N$\mathfrak{I}$ and capturing the intermediate logics obtained by extending intuitionistic logic by any set of Hilbert axioms of the form

$$\mathcal{A}_{\mathtt{i}} = (F_1 \to G_1) \vee \ldots \vee (F_m \to G_m) \tag{6.1}$$

where for every $F_i \neq \top$, $F_i = G_j$ for some j and F_i is not repeated. Notice that the method introduced in Section 3.3.2 enables us to automatically extract higher-level natural deduction rules for the axioms in this shape from hypersequent rules. The subscript L in $\lambda_{\|\mathrm{L}}$ ranges over the intermediate logics that can be defined as extensions of intuitionistic logic by axioms of the form $\mathcal{A}_{\mathtt{i}}$.

Definition 6.1.1 (Terms of $\lambda_{\|\mathrm{L}}$)**.** The terms of $\lambda_{\|\mathrm{L}}$ are defined by the rules in Table 6.1.

Remark 6.1. Notice that, according to the typing rules, channel variables can only occur as applied variables. Thus, they cannot be considered as standalone terms, unlike intuitionistic variables.

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, $z^A, \ldots$ 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 $a^A, b^A, c^A, \ldots$ and represent a *private* communication channel between the parallel processes.

The free and bound variables of a proof term are defined as usual and for the new term ${}_a(u_1 \parallel \ldots \parallel u_m)$, all the free occurrences of a in $u_1, \ldots, u_m$ are bound in ${}_a(u_1 \parallel \ldots \parallel u_m)$.

Remark 6.2. If $\Gamma = y_1 : A_1, \ldots, y_n : A_n$ and all free variables of a proof term $t : A$ are in $y_1, \ldots, y_n$, we write $\Gamma \vdash t : A$. From the logical point of view, t represents a natural deduction derivation of A from the hypotheses $A_1, \ldots, A_n$ if we interpret the applications of the rule below on the left as applications of the rule below on the right

$$\frac{u : A}{a^{A \to B}\, u : B} \qquad\qquad \frac{A \to B \quad A}{B}$$

$$x^A : A \text{ for } x \text{ intuitionistic variable} \qquad \frac{t_1 : A \quad t_2 : A}{t_1 \parallel t_2 : A}\ \mathit{contr}$$

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

$$\frac{\begin{array}{c}[x^A : A]\\ \vdots\\ u : B\end{array}}{\lambda x^A u : A \to B} \qquad \frac{t : A \to B \quad u : A}{tu : B} \qquad \frac{u : \bot}{u\,\mathsf{efq}_P : P}\ \text{with } P \text{ atomic, } P \neq \bot.$$

$$\frac{u : A}{\iota_0(u) : A \vee B} \qquad \frac{u : B}{\iota_1(u) : A \vee B} \qquad \frac{\begin{array}{ccc} & [x^A : A] & [y^B : B]\\ & \vdots & \vdots\\ u : A \vee B & w_1 : C & w_2 : C\end{array}}{u\,[x^A.w_1, y^B.w_2] : C}$$

$$\frac{u : A}{a^{A\to B}\,u : B} \qquad \frac{\begin{array}{ccccc}[a^{F_1\to G_1} : F_1 \to G_1] & & & & [a^{F_m\to G_m} : F_m \to G_m]\\ \vdots & & & & \vdots\\ u_1 : B & & \dots & & u_m : B\end{array}}{{}_a(u_1 \parallel \dots \parallel u_m) : B}\ (\mathcal{A}_{\mathtt{i}})$$

where $(F_1 \to G_1) \vee \dots \vee (F_m \to G_m)$ is an instance of $\mathcal{A}_{\mathtt{i}}$ and all occurrences of a in u_i for $1 \leq i \leq m$ must be of the form $a^{F_i \to G_i}$

Table 6.1: Type assignments for $\lambda_{\|\mathrm{L}}$.

Equivalently, we can consider the rule $(\mathcal{A}_{\mathtt{i}})$ as a higher-level rule, see Section 3.3.2, and the instances of the rule above on the left as the inferences discharged by $(\mathcal{A}_{\mathtt{i}})$.

If the symbol $\|$ does not occur in it, then t is a *simply typed λ-term* representing an intuitionistic deduction.

Definition 6.1.2 (Simple contexts)**.** A *simple context* $\mathcal{C}[\,]$ is a simply typed λ-term with some fixed variable $[\,]$ occurring exactly once. For any proof term u of the same type of $[\,]$, $\mathcal{C}[u]$ denotes the term obtained replacing $[\,]$ with u in $\mathcal{C}[\,]$, *without renaming of bound variables.*

We generalize below the notion of stack to also include the case distinction and injection operators which correspond to the disjunction rules.

Definition 6.1.3 (Stack)**.** A *stack* is a possibly empty sequence $\sigma = \sigma_1\sigma_2\dots\sigma_n$ such that for every $1 \leq i \leq n$, exactly one of the following holds: either $\sigma_i = t$, with t proof term or $\sigma_i = \pi_j$ with $j \in \{0, 1\}$, or $\sigma_i = [x_1.u_1, x_2.v_2]$ or efq_P for some atom P. Now on we will denote the *empty sequence* with ϵ and with $\xi, \xi', \xi_1, \xi_2, \dots$ the stacks of length 1. If t is a proof term, as usual $t\,\sigma$ denotes the term $(((t\,\xi_1)\,\xi_2)\dots\xi_n)$.

Definition 6.1.4 (Case-free)**.** A stack $\sigma = \xi_1\xi_2\dots\xi_n$ is *case-free* if ξ_i is not of the form $[z_1.w_1, z_2.w_2]$ for any $i \in \{1, \dots, n\}$.

Definition 6.1.5 (Parallel form)**.** A *parallel form* is defined inductively as follows: a simply typed λ-term is a parallel form; if $u_1, \ldots, u_m$ are parallel forms, then both ${}_a(u_1 \parallel \ldots \parallel u_m)$ and $u_1 \parallel \ldots \parallel u_m$ are parallel forms.

6.1.1 Reduction rules of $\lambda_{\parallel \mathrm{L}}$

Although the type assignment rules of $\lambda_{\parallel \mathrm{L}}$ and $\lambda_{\parallel}$ are similar, their reduction rules are very different. The reductions of $\lambda_{\parallel \mathrm{L}}$ are based on the notion of value, which captures the idea of a message that has to be transmitted because it has terminated its internal computations but will generate new computation when it is plugged in a new computational environment. This is exactly what we need in order to obtain an analytic normal form. Indeed, if we use all channels that are applied to values, then we trigger all cross reductions that generate redexes. Furthermore this is perfectly sensible from a computational perspective: we trigger the communications that transmit messages with operational content but we do not start a communication only to send empty objects like variables.

Such an approach is very well-suited and quite effective, but still not enough for the subformula property. Indeed, the type of a channel occurrence $a\,t$ does not only depend on its argument t, it globally depends on the term ${}_a(\ldots \parallel \ldots \parallel \ldots)$ in which it is bound. Hence when we reduce some occurrences of a channel because they are applied to values, we need to make sure not to leave other occurrences of the same channel around: their type as well is too complex even if they are not applied to values. There is an easy solution to this problem: when we trigger a communication, we force the term to exhaust all occurrences of the communicating channel. Formally, we distinguish between active and non-active channels. If an occurrence of a channel is applied to a value, we activate the whole channel; when a channel is active, all its occurrences transmit their messages. Finally, a communication will only take place through an active channel.

Definition 6.1.6 (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 ${}_a(u_1 \parallel \ldots \parallel u_m)$ is called an *active session*, whenever a is active.

Our notion of value is defined in order to cover all terms that might directly or indirectly trigger new computation if transmitted to another process. Tuples have a special rôle in the definition of values because they are often used in $\lambda_{\parallel \mathrm{L}}$ as containers of messages. For example, cross reductions introduce new tuples – see Table 6.2 for the definition of these reductions and notice in particular the channel applications $b_i\langle \boldsymbol{y}_i \rangle$. Thus, if we considered all tuples as values with computational content, we would not be able to show that, as the normalization proceeds, communications transmit simpler and simpler values.

Definition 6.1.7 (Value)**.** A *value* is a term of the form $\langle t_1, \ldots, t_n \rangle$, where for some $1 \leq i \leq n$, $t_i = \lambda x\, s$, $t_i = \iota_i(s)$, $t_i = t\,\mathsf{efq}_P$, $t_i = t[x.u, y.v]$ or $t_i = a\sigma$ for an active channel a.

We codify now the idea that a channel can be activated whenever it is applied to at least one value.

Definition 6.1.8 (Activable channel)**.** We say that a is an *activable channel in* u if av occurs in u for some value v.

Notice that there is no circularity between Definitions 6.1.6, 6.1.7 and 6.1.8. Indeed, the classes of active and inactive channels simply constitute a partition of the syntactic class of channels, the notion of value depends on this partition, and the notion of activable channel depends on the notion of value. Nonetheless, the partition of the class of channels into active and inactive is arbitrary and does not depend on the notions of value and activable channel. The notion of value, in turn, does not depend on that of activable channel.

The reduction rules for $\lambda_{\|\mathrm{L}}$-terms are shown in Table 6.2.

Activation reductions These reductions enable us to transform a non-active channel which is applied to a value into an active channel. Only active channels can be used for communications.

Basic cross reductions These reductions implement a simple communication of a term t without dependencies with its original computational environment

$$\begin{aligned} &{}_a(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[a^{F_i \to G_i}\, t] \parallel \ldots \parallel \mathcal{C}_j[a^{F_j \to G_j}\, u] \parallel \ldots \parallel \mathcal{C}_m) \;\mapsto_{\mathrm{L}} \\ &{}_a(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[a^{F_i \to G_j}\, t] \parallel \ldots \parallel \mathcal{C}_j[t] \parallel \ldots \parallel \mathcal{C}_m) \end{aligned}$$

While the sender $\mathcal{C}_i[a_i t]$ is unchanged, $\mathcal{C}_j[a_j\, u]$ receives the message and becomes $\mathcal{C}_j[t]$. Here, unlike in λ_{Cl} cross reductions, only one receiving channel can be used for each communication. Indeed, if some occurrences of a_i are nested, a communication using all of them would break subject reduction.

Cross reductions In order to obtain normal form proofs that enjoy the subformula property we cannot only transmit terms that do not have dependencies with their computational environment; as in λ_{Cl} and λ_{G}, we also need to transmit messages that depend on the computational environment of the sender process. For example, a message may contain free variables which are locally bound in the process and therefore cannot be transmitted without special care, otherwise new free variables would be generated in the whole term. For a concrete example consider the term

$$ {}_a(\; w\,(a\,(\lambda x\, xt)) \parallel {}_b(\lambda z\, z(b(uz)\pi_0) \parallel av(\lambda y\, b\langle s, y\rangle))\;) \tag{6.2}$$

As explained for λ_{Cl}, we would like to allow the active channel b to transmit before the channel a, which might not be ready to transmit yet. There is no reason indeed to wait for the transmission of the value t of y from the first to the third process before using

Intuitionistic reductions (application, projection and injection)

$$(\lambda x^A\, u)t \mapsto_{\mathrm{L}} u[t/x^A] \qquad \langle u_0, u_1\rangle\, \pi_i \mapsto_{\mathrm{L}} u_i \text{ for } i \in \{0,1\} \qquad \iota_i(t)[x_0.u_0, x_1.u_1] \mapsto_{\mathrm{L}} u_i[t/x_i]$$

Case distinction permutations

$$t[x_0.u_0, x_1.u_1]\xi \mapsto_{\mathrm{L}} t[x_0.u_0\xi, x_1.u_1\xi] \text{ if } \xi \text{ is a one-element stack}$$

Parallel operator permutations

$${}_a(u_1 \parallel \ldots \parallel u_m)\,\xi \mapsto_{\mathrm{L}} {}_a(u_1\xi \parallel \ldots \parallel u_m\xi), \text{ if } \xi \text{ is a one-element stack and } a \text{ does not occur free in } \xi$$

$$w_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_a(wu_1 \parallel \ldots \parallel wu_m), \text{ if } a \text{ does not occur free in } w$$

$$\lambda x^A\, {}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_a(\lambda x^A u_1 \parallel \ldots \parallel \lambda x^A u_m)$$

$$\iota_i({}_a(u_1 \parallel \ldots \parallel u_m)) \mapsto_{\mathrm{L}} {}_a(\iota_i(u_1) \parallel \ldots \parallel \iota_i(u_m))$$

$$\langle {}_a(u_1 \parallel \ldots \parallel u_m),\, w\rangle \mapsto_{\mathrm{L}} {}_a(\langle u_1, w\rangle \parallel \ldots \parallel \langle u_m, w\rangle), \text{ if } a \text{ does not occur free in } w$$

$$\langle w,\, {}_a(u_1 \ \ldots \parallel u_m)\rangle \mapsto_{\mathrm{L}} {}_a(\langle w, u_1\rangle \parallel \ldots \parallel \langle w, u_m\rangle), \text{ if } a \text{ does not occur free in } w$$

$${}_a(u_1 \parallel \ldots \parallel {}_b(w_1 \parallel \ldots \parallel w_n) \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_b({}_a(u_1 \parallel \ldots \parallel w_1 \ldots \parallel u_m) \ldots \parallel {}_a(u_1 \parallel \ldots \parallel w_n \ldots \parallel u_m))$$

if $u_1, \ldots, u_m$ and ${}_b(w_1 \parallel \ldots \parallel w_n)$ do not contain active sessions

Communication reductions

Activation reductions ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_b(u_1[b/a] \parallel \ldots \parallel u_m[b/a])$
where a is not active, b is a fresh *active* variable, and there is some occurrence of a in some u_i of the form aw, for a value w.

Basic cross reductions

$${}_a(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[a^{F_i \to G_i}\, t] \parallel \ldots \parallel \mathcal{C}_j[a^{F_j \to G_j}\, u] \parallel \ldots \parallel \mathcal{C}_m) \mapsto_{\mathrm{L}} {}_a(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[a^{F_i \to G_j}\, t] \parallel \ldots \parallel \mathcal{C}_j[t] \parallel \ldots \parallel \mathcal{C}_m) \quad \text{for } F_i = G_j$$

$${}_a(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[a^{F_i \to G_i}\, u] \parallel \ldots \parallel \mathcal{C}_j[a^{F_j \to G_j}\, t] \parallel \ldots \parallel \mathcal{C}_m) \mapsto_{\mathrm{L}} {}_a(\mathcal{C}_1 \parallel \ldots \parallel \mathcal{C}_i[t] \parallel \ldots \parallel \mathcal{C}_j[a^{F_j \to G_j}\, t] \parallel \ldots \parallel \mathcal{C}_m) \quad \text{for } F_j = G_i$$

where a is active, the displayed occurrence of a rightmost in the simply typed λ-terms $\mathcal{C}_i[a^{F_i \to G_i} t]$ and $\mathcal{C}_j[a^{F_j \to G_j} u]$, $1 \le i < j \le m$, $\mathcal{C}_i, \mathcal{C}_j$ are simple contexts and t is closed.

Simplification reductions ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} u_{j_1} \parallel \ldots \parallel u_{j_n}$ for $1 \le j_1 < \ldots < j_n \le m$ if a does not occur in $u_{j_1}, \ldots, u_{j_n}$

Cross reductions

$${}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m]) \mapsto_{\mathrm{L}} {}_b(s_1 \parallel \ldots \parallel s_m)$$

where a is active, $\mathcal{C}_j[a^{F_j \to G_j}\, t_j]$ for $1 \le j \le m$ are simply typed λ-terms; the displayed $a^{F_j \to G_j}$ is rightmost in each of them; b is fresh; for $1 \le i \le m$, we define

$$s_i = \begin{cases} {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_i[t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i\rangle / \boldsymbol{y}_j}] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m]) & \text{if } G_i \neq \bot \\ {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i\rangle] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m]) & \text{if } G_i = \bot \end{cases}$$

where $F_j = G_i$ in the axiom schema if $G_i \neq \bot$; $\boldsymbol{y}_z$ for $1 \le z \le m$ is the sequence of the free variables of t_z bound in $\mathcal{C}_z[a^{F_z \to G_z}\, t_z]$.

Table 6.2: Reduction Rules for $\lambda_{\parallel \mathrm{L}}$ where $\mathcal{A}_{\mathtt{i}} = (F_1 \to G_1) \vee \ldots \vee (F_m \to G_m)$

the channel b, provided that we have a way to transmit t to the second process when it

becomes available. In order to do so, we can use the general version of the cross reduction. From the logical point of view, this reduction enables us to show that the subformula property holds for our natural deduction. From a computational point of view, cross reductions implement communications with an additional mechanism for handling the migration of open functions and their closures, just as in λ_{Cl} and λ_{G}. The reduction has this shape

$$ {}_a(\mathcal{C}_1[a^{F_1\to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m\to G_m}\, t_m]) \;\mapsto_{\mathrm{L}}\; {}_b(s_1 \parallel \ldots \parallel s_m) $$

where for $1 \le i \le m$

$$ s_i = \begin{cases} {}_a(\mathcal{C}_1[a^{F_1\to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[t_j^{\,b^{B_i\to B_j}\langle \boldsymbol{y}_i\rangle/\boldsymbol{y}_j}] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m\to G_m}\, t_m]) & \text{if } G_i \neq \bot \\ {}_a(\mathcal{C}_1[a^{G_1\to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[b^{B_i\to \bot}\langle \boldsymbol{y}_i\rangle] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m\to G_m}\, t_m]) & \text{if } G_i = \bot \end{cases} $$

In this reduction, each process $\mathcal{C}_k[a^{F_k\to G_k}\, t_k]$ transmits its message t_k to the term $\mathcal{C}_l[a^{F_l\to G_l}\, t_l]$ such that $F_k = G_l$, which is the designated receiver of the message. If $G_k = F_n$, our sender $\mathcal{C}_k[a^{F_k\to G_k}\, t_k]$ will also receive a message t_n from another process, and will reduce to $\mathcal{C}_k[t_n^{\,b^{B_k\to B_n}\langle \boldsymbol{y}_k\rangle/\boldsymbol{y}_n}]$. If this is the case, the free variables $\boldsymbol{y}_n$ of the message t_n are going to be replaced by the application of the fresh communication channel $b^{B_k\to B_n}$, which has the task to receive the values of $\boldsymbol{y}_n$ when they will be forwarded. If, on the other hand, $G_k = \bot$ and the sender $\mathcal{C}_k[a^{F_k\to G_k}\, t_k]$ is not a receiver, then it will reduce into $\mathcal{C}_k[b^{B_k\to \bot}\langle \boldsymbol{y}_k\rangle]$ since there is no incoming message. In both cases, the applications $b\langle \boldsymbol{y}_k\rangle$ will be used to forward the values of the variables $\boldsymbol{y}_k$ if and when they will be available. Technically, the redex reduces to the term ${}_b(s_1 \parallel \ldots \parallel s_m)$ where s_p for $p \in \{1, \ldots, m\}$ is a copy of the original redex and contains only one process of the form $\mathcal{C}_k[t_n^{\,b^{B_k\to B_n}\langle \boldsymbol{y}_k\rangle/\boldsymbol{y}_n}]$ or $\mathcal{C}_k[b^{B_k\to \bot}\langle \boldsymbol{y}_k\rangle]$ resulting from the communication. Intuitively, this term ${}_b(s_1 \parallel \ldots \parallel s_m)$ in its entirety encodes the results of all possible communications by the channel a. The unnecessary duplicates will be reduced later on by reductions of the form ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} u_{j_1} \parallel \ldots \parallel u_{j_n}$. Notice in particular that if some term $b^{B_k\to B_n}\langle \boldsymbol{y}_k\rangle$ does not occur because the message t_n does not have free variables $\boldsymbol{y}_n$ in $\mathcal{C}_k[t_n^{\,b^{B_k\to B_n}\langle \boldsymbol{y}_k\rangle/\boldsymbol{y}_n}]$, then a simplification reduction is possible. Intuitively this means that the process $\mathcal{C}_k[t_n^{\,b^{B_k\to B_n}\langle \boldsymbol{y}_k\rangle/\boldsymbol{y}_n}]$ does not need the channel b and hence we can eliminate it immediately. There will be no term then that needs the values of the variables $\boldsymbol{y}_k$ to be forwarded.

This general reduction is inevitably complex, but it can be used in a relatively simple way in practice. As an example, let us consider again the term (6.2) above. One can directly use the channel b to send the term $\langle s, y\rangle$ to the central term without waiting for the channel a to transmit its message and fill y in $b\langle s, y\rangle$ with t. Technically, the reduction is

$$ \begin{aligned} & {}_a(\ w\ (a\,(\lambda x\, x t)) \parallel {}_b(\lambda z\, z(b(uz)\pi_0) \parallel av(\lambda y\, b\langle s, y\rangle))\ ) \\ \mapsto_{\mathtt{p}}\ & {}_a(\ w\ (a(\lambda x\, x\, t)) \parallel {}_c(\ {}_b(\lambda z\, z(\langle s, cz\rangle\pi_0) \parallel av(\lambda y\, b\langle s, y\rangle)) \parallel {}_b(\lambda z\, z(b(uz)\pi_0) \parallel av(\lambda y\, u(cy))))\ ) \\ \mapsto_{\mathtt{p}}\ & {}_a(\ w\ (a(\lambda x\, x\, t)) \parallel {}_c(\ {}_b(\lambda z\, zs \parallel av(\lambda y\, b\langle s, y\rangle)) \parallel {}_b(\lambda z\, z(b(uz)\pi_0) \parallel av(\lambda y\, u(cy))))\ ) \end{aligned} $$

$$\mapsto_{\mathtt{p}}^{*}\ {}_a(\ w\ (a(\lambda x\, x\, t))\parallel {}_c(\lambda z\, zs \parallel av(\lambda y\, u(cy)))\ )$$
$$\mapsto_{\mathtt{p}}\ {}_a(\ w\ (a(\lambda x\, x\, t))\parallel \lambda z\, zs)\ \mapsto_{\mathtt{p}} \lambda z\, zs$$

We only apply a cross reduction to a term ${}_a(u_1 \parallel \ldots \parallel u_m)$ when the channel a is *active*. Ultimately, a channel can be activated only if one of its occurrences is directly applied to a value or may receive a value to be transmitted sometimes in the future. Intuitively, an active channel has or will have an argument that needs to be transmitted because it might produce new computations in the receiving process. This activation condition would be natural in a call-by-value reduction strategy, but here we are not transmitting messages according to a call-by-value policy just for the sake of it. An activation condition is indeed *necessary*, because unrestricted cross reductions do not always terminate. For example, reducing

$${}_a(\lambda y^{B}\, a^{B\to B}\, y \parallel x^{(C\to C)\to B\to B}\, (\lambda z^{C}\, a^{C\to C}\, z))$$

as follows generates a loop:

$$\mapsto_{\mathtt{p}}\ {}_b({}_a((\lambda y\, b\, y \parallel x\, (\lambda z\, az))) \parallel {}_a((\lambda y\, a\, y \parallel x\, \lambda z\, bz)))$$

$$\mapsto_{\mathtt{p}}^{*}\ {}_b(\lambda y\, b\, y \parallel x\, \lambda z\, bz)\ \mapsto_{\mathtt{p}} \ldots$$

For a proof-theoretic view, consider the application of $(\mathcal{A}_{\mathtt{i}})$ (below left), in which all Γ_i for $1 \le i \le m$ are discharged between G_i and B. It reduces by cross reduction to the derivation below right:

$$\dfrac{\dfrac{\dfrac{\Gamma_1}{\alpha_1}}{\dfrac{F_1}{G_1}}\ \ldots\ \dfrac{\dfrac{\Gamma_m}{\alpha_m}}{\dfrac{F_m}{G_m}}}{\dfrac{\vdots\qquad\qquad\vdots}{\dfrac{B\quad\ldots\quad B}{B}\,*}} \qquad \mapsto_{\mathrm{L}} \qquad \dfrac{\delta_1\quad\ldots\quad\delta_m}{B}\,{**}$$

such that for $1 \le i \le m$ the derivation δ_i is

$$\dfrac{\ldots\quad\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_i}{\bigwedge\Gamma_i}\wedge i}{\bigwedge\Gamma_j}}{\alpha_j}\wedge e}{\dfrac{G_i}{\dfrac{\vdots}{\dfrac{B\quad\ldots\quad B}{B}\,*}}}\quad\ldots\quad\ldots\quad B}{}$$

in which α_j is the derivation of the premise $F_j = G_i$ associated by $\mathcal{A}_{\mathtt{i}}$ to G_i and a double inference line denotes a derivation of its conclusion using the named rule possibly many times.

Theorem 6.1.1 (Subject reduction)**.** *If $t : A$ and $t \mapsto_{\mathrm{L}} u$, then $u : A$ and all the free variables of u appear among those of t.*

Proof. See Theorem 2.4.3 for intuitionistic reductions. Since, as usual, the argument for simplification reductions is trivial and that for basic cross reductions is just a simpler version of that for cross reductions, we only consider the latter case. Consider the reduction

$$_a(\mathcal{C}_1[a^{F_1\to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m\to G_m}\, t_m]) \;\mapsto_{\mathrm{L}}\; _b(s_1 \parallel \ldots \parallel s_m)$$

Suppose that $G_i \neq \bot$ for $1 \leq i \leq m$ and hence s_i is

$$_a(\mathcal{C}_1[a^{F_1\to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_i[t_j^{b^{B_i\to B_j}\langle \boldsymbol{y}_i\rangle/\boldsymbol{y}_j}] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m\to G_m}\, t_m])$$

Assuming that $\langle \boldsymbol{y}_i\rangle : B_i$, the terms $b^{B_i\to B_j}\langle \boldsymbol{y}_i\rangle$ are correct. Hence the term $t_j^{b^{B_i\to B_j}\langle \boldsymbol{y}_i\rangle/\boldsymbol{y}_j}$, by Definition 4.1.6, is correct as well. Now, some of the assumptions of the reduction rule are that $t_j^{b^{B_i\to B_j}\langle \boldsymbol{y}_i\rangle/\boldsymbol{y}_j}$ has the same type as $a^{F_i\to G_i}\, t_i$; that $\boldsymbol{y}_i$ for $1 \leq i \leq m$ is the sequence of the free variables of t_i which are bound in $\mathcal{C}_i[a^{F_i\to G_i}\, t_i]$. Hence, by construction, all the variables $\boldsymbol{y}_i$ are bound in each $\mathcal{C}_i[t_j^{b^{B_i\to B_j}\langle \boldsymbol{y}_i\rangle/\boldsymbol{y}_j}]$. Since, moreover, a is rightmost in each $\mathcal{C}_i[a^{F_i\to G_i}\, t_i]$ and b is fresh, no new free variable is created.

Suppose, on the other hand, that $G_i = \bot$ for $1 \leq i \leq m$ and hence s_i is

$$_a(\mathcal{C}_1[a^{F_1\to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_i[b^{B_i\to \bot}\langle \boldsymbol{y}_i\rangle] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m\to G_m}\, t_m])$$

Assuming that $\langle \boldsymbol{y}_i\rangle : B_i$, the term $b^{B_i\to \bot}\langle \boldsymbol{y}_i\rangle$ is a correct term of type $\bot$. Now, since $G_i = \bot$, the term $b^{B_i\to \bot}\langle \boldsymbol{y}_i\rangle$ has the same type as $a^{F_i\to G_i}\, t_i$. Since $\boldsymbol{y}_i$ for $1 \leq i \leq m$ is the sequence of the free variables of t_i which are bound in $\mathcal{C}_i[a^{F_i\to G_i}\, t_i]$ and since b is fresh, no new free variable is created.

The argument for permutations is a trivial generalization of that for Theorem 4.1.3. □

Definition 6.1.9 (Normal forms)**.** We define NF_{L} to be the set of normal $\lambda_{\|\mathrm{L}}$-terms as defined in Definition 2.4.2.

Remark 6.3 (**Ambient calculus and cross reductions**)**.** Ambient calculus by Cardelli and Gordon, see e.g. [CG00], is a formalism in which processes are represented as ambients in which the computation is carried out. Ambients' structure is similar to the structure of processes in process calculi, but instead of being able to communicate, ambients can enter, exit, or dissolve the boundaries of other ambients. Even though the issues addressed by ambient calculus seem close to those that motivate the introduction of cross reductions in $\lambda_{\|\mathrm{L}}$, the operations of ambients are assumed not to break any computational dependence, while cross reductions are precisely meant to handle changes of environment which violate computational dependencies by restoring such dependencies.

6.2 The normalization theorem

We prove the normalization theorem for $\lambda_{\|\mathrm{L}}$: every proof term of $\lambda_{\|\mathrm{L}}$ reduces in a finite number of steps to a normal form. By subject reduction, this implies that the corresponding natural deduction proofs do normalize.

The idea behind our normalization strategy is quite intuitive. We start from any term and reduce it in parallel form by Proposition 6.2.2. 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 all messages and the processes to extract information from the received messages. Technically, we perform all cross reductions combined with the generated projections and case distinction 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 channels that need 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. Hence the count of active sessions might increase forever and the communication phase never terminate.

We avoid the first loop by exploiting the complexity of the exchanged messages. Since messages are *values*, we shall define a notion of *value complexity* (Definition 6.2.4), which will simultaneously ensure that: (i) after reducing an intuitionistic redex which is not a projection or a case distinction permutation, the new active sessions only transmit messages with value complexity smaller than the complexity of such redex; (ii) after transmitting a message, all newly generated intuitionistic redexes do not have complexity greater than the value complexity of such message. Proposition 6.2.6) will settle the matter, but in turn requires a series of preparatory lemmas. Namely, we shall study in Lemma 6.2.5 and Lemma 6.2.6 how arbitrary substitutions affect the value complexity of a term; then we shall determine in Lemma 6.2.8 and Lemma 6.2.7 how case reductions impact the value complexity.

We prove that the second loop is not possible by showing in Lemma 6.2.10 that the transmission of messages, during the communication phase, cannot produce new active sessions. Intuitively, the newly generated channels and the old duplicated ones are *frozen*, in the sense that only the reduction of an intuitionistic redex can activate them and in

doing it will generate communication redexes with smaller complexity than the redex itself.

For clarity, we define here a recursive normalization algorithm that represents the constructive content of the normalization proof. The master reduction strategy consists of three phases: one in which we reduce all possible intuitionistic redexes, one in which we activate all possible sessions, and one during which we exploit the reduction relation $\succ_{\mathrm{L}}$ defined below, whose purpose 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 λ-terms and finally apply the cross reductions followed by projections and case permutations.

Definition 6.2.1 (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

$$t \succ_{\mathrm{L}} t'$$

whenever t' has been obtained from t by applying one of the following to ${}_a(u_1 \parallel \ldots \parallel u_m)$:

1. a permutation reduction

$$\begin{array}{l} {}_a(u_1 \parallel \ldots \parallel {}_b(w_1 \parallel \ldots \parallel w_n) \ldots \parallel u_m) \;\mapsto_{\mathrm{L}} \\ {}_b({}_a(u_1 \parallel \ldots \parallel w_1 \ldots \parallel u_m) \parallel \ldots \parallel {}_a(u_1 \parallel \ldots \parallel w_n \ldots \parallel u_m)) \end{array}$$

   if $u_i = {}_b(w_1 \parallel \ldots \parallel w_n)$ 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 distinction permutations inside the newly generated simply typed λ-terms;

3. a simplification reduction ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} u_{j_1} \parallel \ldots \parallel u_{j_n}$, for $1 \leq j_1 < \ldots < j_n \leq m$, if a does not occur in $u_{j_1}, \ldots, u_{j_n}$

Definition 6.2.2 (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 6.2.1) to u, then we go to step 1.

We start by 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. For terms of the form λxu and $\iota_i(u)$ – which are values in the usual sense – such complexity is defined on the type of the term. For pairs $\langle u, v\rangle$ and case distinctions $t[x.u, y.v]$ – which we consider containers for what is usually taken to be a value – we recursively pick 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 normalization argument would completely break down when new channels are generated during cross reductions: their type can be much more complex than the type of the original channel and any hope of finding a decreasing measure would be shattered.

Definition 6.2.3 (Complexity of a type)**.** The complexity of a type T is recursively defined as follows:

- if $T = \bot$ or $T = P$ for a propositional variable P, then the complexity of T is 0;
- if $T = A_1 \to A_2$, $T = A_1 \wedge A_2$ or $T = A_1 \vee A_2$ and the complexity of A_i is n_i, then the complexity of T is $n_1 + n_2 + 1$.

Definition 6.2.4 (Value complexity)**.** For any simply typed λ-term $s : T$, the value complexity of s is defined as the first case that applies among the following:

- if $s = \lambda xu$, $s = \iota_i(u)$, then the value complexity of s is the complexity of its type T;
- if $s = \langle u, v\rangle$, 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 σ 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 supposed to be those terms that can either generate an intuitionistic redex when plugged into another term or become a term with that capability, like an active channel acting as the endpoint of a transmission.

The value complexity of a term, as expected, never surpass the complexity of its type.

Proposition 6.2.1. *Let $u : T$ be any simply typed λ-term. Then the value complexity of u is not greater than 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 λxw, $\iota_i(w)$, 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 $T = T_1 \wedge T_2$, and we are done.
- If u is of the form $v_0[z_1.v_1, z_2.v_2]$ then, by induction hypothesis, the value complexities of v_1 and v_2 are at most the complexity of T, and we are done.
- In all other cases, the value complexity of u is 0, which trivially satisfies the thesis.

□

The complexity of an intuitionistic redex $t\xi$ is defined as the value complexity of t.

Definition 6.2.5 (Complexity of the intuitionistic redexes)**.** Let r be an intuitionistic redex. The complexity of r is defined as follows:

- If $r = (\lambda xu)t$, then the complexity of r is the type of λxu.
- If $r = \iota_i(t)[x.u, y.v]$, then the complexity of r is the type of $\iota_i(t)$.
- if $r = \langle u, v\rangle\pi_i$ then the complexity of r is the value complexity of $\langle u, v\rangle$.
- if $r = t[x.u, y.v]\xi$, then the complexity of r is the value complexity of $t[x.u, y.v]$.

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

Definition 6.2.6 (Complexity of the communication redexes)**.**

- The *complexity of a channel occurrence* $a\langle t_1, \ldots, t_n\rangle$ of a channel a is the value complexity of $\langle t_1, \ldots, t_n\rangle$.
- The *complexity of a communication redex* ${}_a(u_1 \parallel \ldots \parallel u_m)$is the maximum among the complexities of the occurrences of a in $u_1, \ldots, u_m$.
- The *complexity of a permutation redex* ${}_a(u_1 \parallel \ldots \parallel {}_b(w_1 \parallel \ldots \parallel w_n) \ldots \parallel u_m)$ 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 6.2.7. We distinguish two groups of redexes:

1. Group 1: Application and injection redexes.
2. Group 2: Communication redexes, projection redexes and case distinction permutation redexes.

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

Proposition 6.2.2 (Parallel form)**.** *Let $t : A$ be any term. Then $t \mapsto_{\mathrm{L}}^{*} t'$, where t' is a parallel form.*

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

- t is a variable x. Trivial.
- $t = \lambda x\, u$. By induction hypothesis, $u \rightrightarrows^{*} u_1 \parallel u_2 \parallel \ldots \parallel u_n$ and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying several permutations we obtain

  $$t \rightrightarrows^{*} \lambda x\, u_1 \parallel \lambda x\, u_2 \parallel \ldots \parallel \lambda x\, u_n$$

  which is the thesis.
- $t = u\, v$. By induction hypothesis,

  $$u \rightrightarrows^{*} u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

  $$v \rightrightarrows^{*} v_1 \parallel v_2 \parallel \ldots \parallel v_m$$

  and each term u_i and v_i, for $1 \leq i \leq n, m$, is a simply typed λ-term. Applying several permutations we obtain

  $$\begin{aligned} t &\rightrightarrows^{*} (u_1 \parallel u_2 \parallel \ldots \parallel u_n)\, v \\ &\rightrightarrows^{*} u_1\, v \parallel u_2\, v \parallel \ldots \parallel u_n\, v \\ &\rightrightarrows^{*} u_1\, v_1 \parallel u_1\, v_2 \parallel \ldots \parallel u_1\, v_m \parallel \ldots \\ &\qquad \ldots \parallel u_n\, v_1 \parallel u_n\, v_2 \parallel \ldots \parallel u_n\, v_m \end{aligned}$$

- $t = \langle u, v \rangle$. By induction hypothesis,

  $$u \rightrightarrows^{*} u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

  $$v \rightrightarrows^{*} v_1 \parallel v_2 \parallel \ldots \parallel v_m$$

  and each term u_i and v_i, for $1 \leq i \leq n, m$, is a simply typed λ-term. Applying several permutations we obtain

  $$\begin{aligned} t &\rightrightarrows^{*} \langle u_1 \parallel u_2 \parallel \ldots \parallel u_n,\, v \rangle \\ &\rightrightarrows^{*} \langle u_1, v \rangle \parallel \langle u_2, v \rangle \parallel \ldots \parallel \langle u_n, v \rangle \\ &\rightrightarrows^{*} \langle u_1, v_1 \rangle \parallel \langle u_1, v_2 \rangle \parallel \ldots \parallel \langle u_1, v_m \rangle \parallel \ldots \\ &\qquad \ldots \parallel \langle u_n, v_1 \rangle \parallel \langle u_n, v_2 \rangle \parallel \ldots \\ &\qquad \ldots \parallel \langle u_n, v_m \rangle \end{aligned}$$

- $t = u\,\pi_i$. By induction hypothesis,

$$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying several permutations we obtain

$$t \rightrightarrows^* u_1\,\pi_i \parallel u_2\,\pi_i \parallel \ldots \parallel u_n\,\pi_i.$$

- $t = \iota_i(u)$. By induction hypothesis,

$$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying several permutations we obtain

$$t \rightrightarrows^* \iota_i(u_1) \parallel \iota_i(u_2) \parallel \ldots \parallel \iota_i(u_n).$$

- $t = s[x.u, y.v]$. By induction hypothesis,

$$\begin{aligned} s &\rightrightarrows^* s_1 \parallel s_2 \parallel \ldots \parallel s_n \\ u &\rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_m \\ v &\rightrightarrows^* v_1 \parallel v_2 \parallel \ldots \parallel v_p \end{aligned}$$

and each term s_i for $1 \leq i \leq n$, u_j for $1 \leq j \leq m$, and v_k for $1 \leq k \leq p$ is a simply typed λ-term. Applying several permutations we obtain

$$\begin{aligned} t &\rightrightarrows^* s_1[x.u, y.v] \parallel s_2[x.u, y.v] \parallel \ldots \parallel s_n[x.u, y.v] \\ &\rightrightarrows^* s_1[x.u_1, y.v] \parallel s_1[x.u_2, y.v] \parallel \ldots \parallel s_1[x.u_m, y.v] \parallel \\ &\quad\; s_2[x.u_1, y.v] \parallel s_2[x.u_2, y.v] \parallel \ldots \parallel s_2[x.u_m, y.v] \parallel \\ &\quad\; \ldots \parallel s_n[x.u_1, y.v] \parallel s_n[x.u_2, y.v] \parallel \ldots \parallel s_n[x.u_m, y.v] \\ &\rightrightarrows^* s_1[x.u_1, y.v_1] \parallel s_1[x.u_1, y.v_2] \parallel \ldots \parallel s_1[x.u_1, y.v_p] \parallel \\ &\quad\; s_1[x.u_2, y.v_1] \parallel s_1[x.u_2, y.v_2] \parallel \ldots \parallel s_1[x.u_2, y.v_p] \parallel \\ &\quad\; \ldots \parallel s_1[x.u_m, y.v_1] \parallel s_1[x.u_m, y.v_2] \parallel \ldots \parallel s_1[x.u_m, y.v_p] \\ &\quad\; s_2[x.u_1, y.v_1] \parallel s_2[x.u_1, y.v_2] \parallel \ldots \parallel s_2[x.u_1, y.v_p] \parallel \\ &\quad\; s_2[x.u_2, y.v_1] \parallel s_2[x.u_2, y.v_2] \parallel \ldots \parallel s_2[x.u_2, y.v_p] \parallel \\ &\quad\; \ldots \parallel s_2[x.u_m, y.v_1] \parallel s_2[x.u_m, y.v_2] \parallel \ldots \parallel s_2[x.u_m, y.v_p] \\ &\quad\; s_n[x.u_1, y.v_1] \parallel s_n[x.u_1, y.v_2] \parallel \ldots \parallel s_n[x.u_1, y.v_p] \parallel \\ &\quad\; s_n[x.u_2, y.v_1] \parallel s_n[x.u_2, y.v_2] \parallel \ldots \parallel s_n[x.u_2, y.v_p] \parallel \\ &\quad\; \ldots \parallel s_n[x.u_m, y.v_1] \parallel s_n[x.u_m, y.v_2] \parallel \ldots \parallel s_n[x.u_m, y.v_p]. \end{aligned}$$

- $t = u\,\mathsf{efq}_P$. By induction hypothesis,

$$u \rightrightarrows^* u_1 \parallel u_2 \parallel \ldots \parallel u_n$$

and each term u_i, for $1 \leq i \leq n$, is a simply typed λ-term. Applying several permutations we obtain

$$t \rightrightarrows^* u_1\,\mathsf{efq}_P \parallel u_2\,\mathsf{efq}_P \parallel \ldots \parallel u_n\,\mathsf{efq}_P$$

□

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

Lemma 6.2.3 (Activate!)**.** *Let t be any term in parallel form that does not contain intuitionistic redexes and whose communication redexes have complexity at most τ. 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 τ.*

Proof. The proof is by induction on the number n of subterms of the form ${}_a(u_1 \parallel \ldots \parallel u_m)$ 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 $r = {}_a(v_1 \parallel \ldots \parallel v_m)$. We apply an activation reduction to r and obtain a term r' with $n-1$ subterms of the form ${}_a(u_1 \parallel \ldots \parallel u_m)$ which are not active sessions. In order to apply the induction hypothesis on r', which immediately yields the thesis, we only need to verify that all communication redexes of r' have complexity at most τ.

For this purpose, let c be any channel variable which is bound in r'. Since r' is obtained from r just by renaming the non-active bound channel variable a to an active one b, every occurrence of c in r' is of the form $(c\,t)[b/a]$ for some subterm $c\,t$ of r. Thus $c\,t[b/a] = c[b/a]\langle t_1[b/a], \ldots, t_n[b/a]\rangle$, where each t_i is not a pair. It is enough to show that the value complexity of $t_i[b/a]$ is exactly the value complexity of t_i. We proceed by induction on the size of t_i. We can write $t_i = r\,\sigma$, where σ is a case-free stack. If r is of the form $\lambda x w$, $\langle q_1, q_2\rangle$, $\iota_i(w)$, x, dw, with d channel variable, then the value complexity of $t_i[b/a]$ is the same as that of t_i; note that if $r = \langle q_1, q_2\rangle$, then σ is not empty. If $r = v_0[x_1.v_1, x_2.v_2]$, then σ is empty, otherwise s would contain a permutation redex. Hence, the value complexity of $t_i[b/a]$ is the maximum among the value complexities of $v_1[b/a]$ and $v_2[b/a]$. By induction hypothesis, their value complexities are respectively those of v_1 and v_2 , hence the value complexity of $t_i[b/a]$ is the same as that of t_i, which concludes the proof. □

We show a simple property of the value complexity that we will need later.

Lemma 6.2.4 (Why not 0?)**.** *Let u be any simply typed λ-term and σ 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 u is of the form $(\lambda x.w)\rho$, $\iota_i(w)\rho$, $\langle v_1, v_2\rangle\rho$, $a\rho$ or $x\rho$, where ρ is case-free, then $u\sigma$ has value complexity 0.
- If u is of the form $w\,\mathsf{efq}_P$, then P is atomic, thus σ must be empty, contrary to the assumptions.
- If u is of the form $v_0[z_1.v_1, z_2.v_2]\rho$, with ρ 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.

□

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

Definition 6.2.8 (Simple replacement)**.** By $s\{t/u\}$ we denote a term obtained from s by 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 $w\{v/s\}$ either remains at most as it was before the substitution or becomes exactly the value complexity of v.

Lemma 6.2.5 (The change of value)**.** *Let w, s, v be simply typed λ-terms with value complexity respectively θ, τ, τ'. Then the value complexity of $w\{v/s\}$ is either at most θ or equal to τ'. Moreover, if $\tau' \leq \tau$, then the value complexity of $w\{v/s\}$ is at most θ.*

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 τ'. Moreover, if $\tau' \leq \tau$, since $w = x = s$, we have $\theta = \tau$, thus $\tau' \leq \theta$.
  - $s \neq x$. The value complexity of $w\{v/s\}$ is θ and we are done.
- $w\{v/s\} = \lambda x u\,\{v/s\}$. We have two cases.
  - $\lambda x u\,\{v/s\} = \lambda x(u\{v/s\})$. Then the value complexity of $w\{v/s\}$ is θ and we are done.
  - $\lambda x u\,\{v/s\} = v$. Then the value complexity of $w\{v/s\}$ is τ'. Moreover, if $\tau' \leq \tau$, since $w = \lambda x u = s$, we have $\theta = \tau$, thus $\tau' \leq \theta$.

- $w\{v/s\} = \langle q_1, q_2\rangle\,\{v/s\}$. We have two cases.
  - $\langle q_1, q_2\rangle\,\{v/s\} = \langle q_1\{v/s\}, q_2\{v/s\}\rangle$. Then by induction hypothesis the value complexities of $q_1\{v/s\}$ and $q_2\{v/s\}$ are at most θ or equal to τ'. 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.
  - $\langle q_1, q_2\rangle\,\{v/s\} = v$. Then the value complexity of $w\{v/s\}$ is τ'. Moreover, if $\tau' \leq \tau$, since $w = \langle q_1, q_2\rangle = s$, we have $\theta = \tau$, thus $\tau' \leq \theta$.
- $w\{v/s\} = \iota_i(u)\,\{v/s\}$. We have two cases.
  - $\iota_i(u)\,\{v/s\} = \iota_i(u\{v/s\})$. Then the value complexity of $w\{v/s\}$ is θ and we are done.
  - $\iota_i(u)\,\{v/s\} = v$. Then the value complexity of $w\{v/s\}$ is τ'. Moreover, if $\tau' \leq \tau$, since $w = \iota_i(u) = s$, we have $\theta = \tau$, thus $\tau' \leq \theta$.
-
  - $w\{v/s\} = (v_0[z_1.v_1, z_2.v_2])\rho\{v/s\} = v_0\{v/s\}[z_1.v_1\{v/s\}, z_2.v_2\{v/s\}](\rho\{v/s\})$, where ρ is a case-free stack. If ρ is not empty, then by Lemma 6.2.4, $v_1\{v/s\}\rho\{v/s\}$ and $v_2\{v/s\}\rho\{v/s\}$ have value complexity 0, so $w\{v/s\}$ has value complexity $0 \leq \theta$ and we are done. So assume ρ is empty. By induction hypothesis applied to $v_1\{v/s\}$ and $v_2\{v/s\}$, the value complexity of $w\{v/s\}$ is at most θ or equal to τ' 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 θ. Hence, the value complexity of $(v_0[z_1.v_1, z_2.v_2])\rho\{v/s\}$ is at most θ and we are done.
  - $w\{v/s\} = (v_0[z_1.v_1, z_2.v_2]\rho)\{v/s\} = v\rho_j\{v/s\}\ldots\rho_n\{v/s\}$, where $\rho = \rho_1\ldots\rho_n$ is a case-free stack and $1 \leq j \leq n$. If $\rho_j\ldots\rho_n$ is not empty, then by Lemma 6.2.4, the value complexity of $w\{v/s\}$ is $0 \leq \theta$, and we are done. So assume $\rho_j\ldots\rho_n$ is empty. Then the value complexity of $w\{v/s\}$ is τ'. Moreover, if $\tau' \leq \tau$, since it must be $s = v_0[z_1.v_1, z_2.v_2]$, we have that the value complexity of $w\{v/s\} = v$ is $\tau' \leq \tau = \theta$.
- In all other cases, $w\{v/s\} = (r\,\rho)\{v/s\}$ where ρ is a case-free non-empty stack and r is of the form λxu, $\langle q_1, q_2\rangle$, $\iota_i(u)$, x, or au. We distinguish three cases.
  - $(r\,\rho)\{v/s\} = r\{v/s\}\rho\{v/s\}$ and $r \neq s$. Then by Lemma 6.2.4 the value complexity of $w\{v/s\}$ is $0 \leq \theta$ and we are done.
  - $(r\,\rho)\{v/s\} = v\rho_j\{v/s\}\ldots\rho_n\{v/s\}$, with $\rho = \rho_1\ldots\rho_n$ and $1 \leq j \leq n$. Then by Lemma 6.2.4 the value complexity of $w\{v/s\}$ is $0 \leq \theta$ and we are done.
  - $r\rho\{v/s\} = v$. Then the value complexity of $w\{v/s\}$ is τ'. Moreover, if $\tau' \leq \tau$, since $w = r\rho = s$, we have $\theta = \tau$, thus $\tau' \leq \theta$.

□

The following lemma studies how the complexity of redexes in a term change after a simple replacement.

Lemma 6.2.6 (Replace!)**.** *Let u be a term in parallel form, v, s be any simply typed λ-terms, τ be the value complexity of v and τ' be the maximum among the complexities of the channel occurrences in v. Then every redex in $u\{v/s\}$ it 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 τ or is a communication redex of complexity at most τ'.*

Proof. We prove the following stronger statement.

($*$) Every redex and channel occurrence in $u\{v/s\}$ it 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 r or channel occurrence aw of u, or (iii) has complexity τ or τ' or is a communication redex of complexity at most τ'.

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

- $\langle t_1, t_2 \rangle\, \sigma\, \{v/s\}$, where $\sigma = \sigma_1 \dots \sigma_n$ 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 $\langle t_1, t_2 \rangle\, \sigma\{v/s\} = \langle t_1\{v/s\}, t_2\{v/s\} \rangle\, (\sigma\{v/s\})$, all the redexes and channel occurrences that we have to check are in $t_i\{v/s\}, \sigma\{v/s\}$ or, possibly, the head redex. The former are dealt with using the inductive hypothesis. As for the latter, by Lemma 6.2.5, the value complexity of $t_i\{v/s\}$ for $i \in \{1, 2\}$ must be at most the value complexity of t_i or exactly τ, thus either (ii) or (iii) holds. If $\langle t_1, t_2 \rangle\, \sigma\{v/s\} = v\, (\sigma_i\{v/s\}) \dots (\sigma_n\{v/s\})$, then $v\, (\sigma_i\{v/s\})$ could be a new intuitionistic redex, when $v = \lambda y\, w$, $v = \langle w_1, w_2 \rangle$, $v = \iota_i(w)$ or $v = w_0[y_1.w_1, y_2.w_2]$. But the complexity of such a redex is equal to τ.

- $\iota_i(t)\, \sigma\, \{v/s\}$, where $\sigma = \sigma_1 \dots \sigma_n$ is any case-free stack. Obviously σ must be empty since it is case-free. By induction hypothesis, ($*$) holds for $t'\{v/s\}$. If $\iota_i(t)\{v/s\} = \iota_i(t\{v/s\})$, all the redexes and channel occurrences that we have to check are in $t\{v/s\}$ and thus the thesis holds. If $\iota_i(t)\{v/s\} = v$ then (i) holds.

- $w_0[z_1.w_1, z_2.w_2]\sigma\{v/s\}$, where σ is any case-free stack. By induction hypothesis, $(*)$ holds for $w_0\{v/s\}$, $w_1\{v/s\}$, $w_2\{v/s\}$ and $\sigma_i\{v/s\}$ for $1 \leq i \leq n$. If

$$w_0[z_1.w_1, z_2.w_2]\sigma\{v/s\} = \\ w_0\{v/s\}[z_1.w_1\{v/s\}, z_2.w_2\{v/s\}](\rho\{v/s\})$$

  we first observe that by Lemma 6.2.5, the value complexity of $w_0\{v/s\}$ is at most that of w_0 or exactly τ, hence the possible injection or case distinction permutation redex

$$w_0\{v/s\}[z_1.w_1\{v/s\}, z_2.w_2\{v/s\}]$$

  satisfies the thesis. Again by Lemma 6.2.5, the value complexities of $w_1\{v/s\}$ and $w_2\{v/s\}$ are respectively at most that of w_1 and w_2 or exactly τ. Hence the complexity of the possible case distinction permutation redex

$$(w_0[z_1.w_1\{v/s\}, z_2.w_2\{v/s\}])\sigma_1\{v/s\}$$

  is either τ, and we are done, or at most the value complexity of one among w_1, w_2, thus at most the value complexity of the case distinction permutation redex $(w_0[z_1.w_1, z_2.w_2])\sigma_1$ and we are done.

  If $w_0[z_1.w_1, z_2.w_2]\sigma\{v/s\} = v\,(\sigma_i\{v/s\})\ldots(\sigma_n\{v/s\})$, then there could be a new intuitionistic redex, when $v = \lambda y\, q$, $v = \langle q_1, q_2\rangle$, $v = \iota_i(q)$ or $v = q_0[y_1.q_1, y_2.q_2]$. But the complexity of such a redex is τ.

- $x\,\sigma\{v/s\}$, 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. If $x\,\sigma\{v/s\} = v\,(\sigma_i\{v/s\})\ldots(\sigma_n\{v/s\})$, then $v\,(\sigma_i\{v/s\})$ could be an intuitionistic redex, when $v = \lambda y\, w$, $v = \langle w_1, w_2\rangle$, $v = \iota_i(w)$ or $v = w_0[y_1.w_1, y_2.w_2]$. But the complexity of such a redex is equal to τ.

- $a\,t\,\sigma\{v/s\}$, 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 $t\{v/s\}, \sigma_i\{v/s\}$ where $1 \leq i \leq n$.

  If $a\,t\,\sigma\{v/s\} = v\,(\sigma_i\{v/s\})\ldots(\sigma_n\{v/s\})$, then $v\,(\sigma_i\{v/s\})$ could be an intuitionistic redex, when $v = \lambda y\, w$, $v = \langle w_1, w_2\rangle$, $v = \iota_i(w)$ or $v = w_0[y_1.w_1, y_2.w_2]$. But the complexity of such a redex is equal to τ.

  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 $a\,(t\{v/s\})$. By Lemma 6.2.5, the value complexity of $t\{v/s\}$ is at most the value complexity of t or exactly τ, thus either (ii) or (iii) holds.

- ${}_a(t_1 \parallel \ldots \parallel t_m)\{v/s\}$. By induction hypothesis, $(*)$ holds for $t_i\{v/s\}$ where $1 \leq i \leq m$. The only redex in ${}_a(t_1 \parallel \ldots \parallel t_m)\{v/s\}$ and not in some $t_i\{v/s\}$ can be ${}_a(t_1\{v/s\} \parallel \ldots \parallel t_m\{v/s\})$ itself. But the complexity of such redex equals the maximal complexity of the channel occurrences of the form aw occurring in some $t_i\{v/s\}$, hence it is τ, at most τ' or equal to the complexity of ${}_a(t_1 \parallel \ldots \parallel t_m)$.

□

We study now the complexity of the redexes generated by the reduction of an injection redex.

Lemma 6.2.7 (Eliminate the case!)**.** *Let u be a term in parallel form. Then for any redex r in $u\{w_i[t/x_i]/\iota_i(t)[x_1.w_1, x_2.w_2]\}$ of complexity θ, either $\iota_i(t)[x_1.w_1, x_2.w_2]$ has complexity greater than θ; or there is a redex in u of complexity θ which belongs to the same group as r or is a case distinction permutation redex.*

Proof. Let $v = w_i[t/x_i]$ and $s = \iota_i(t)[x_1.w_1, x_2.w_2]$. We prove a stronger statement:

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

The proof is by induction on the size of u and by cases according to the possible shapes of u.

- $(\lambda x\, t')\, \sigma\, \{v/s\}$, where $\sigma = \sigma_1 \dots \sigma_n$ is any case-free stack. By induction hypothesis, $(*)$ holds for $t'\{v/s\}$ and $\sigma_i\{v/s\}$ for $1 \leq i \leq n$. If $(\lambda x\, t')\, \sigma\{v/s\} = (\lambda x\, t'\{v/s\})\, (\sigma\{v/s\})$, all the redexes and channel occurrences that we have to check are in $t'\{v/s\}$, $\sigma\{v/s\}$ or, possibly, the head redex, thus the thesis holds. Since $s \neq (\lambda x\, t')\, \sigma_1 \dots \sigma_j$, there is no other possible case.

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

  If $\langle t_1, t_2\rangle\, \sigma\{v/s\} = \langle t_1\{v/s\}, t_2\{v/s\}\rangle\, (\sigma\{v/s\})$ all the redexes and channel occurrences that we have to check are either in $\sigma\{v/s\}$ or, possibly, the head redex. By Lemma 6.2.5, the value complexity of $w_i[t/x_i]$ is either at most the value complexity of w_i or the value complexity of t. In the first case, the value complexity of $w_i[t/x_i]$ is at most the value complexity of w_i, which is at most the value complexity of $\iota_i(t)[x_1.w_1, x_2.w_2]$. Thus, by Lemma 6.2.5, the value complexities of $t_1\{v/s\}, t_2\{v/s\}$ are at most the value complexities respectively of t_1, t_2, thus the value complexity of $\langle t_1\{v/s\}, t_2\{v/s\}\rangle$, and hence that of the possible head redex, is at most the value complexity of $\langle t_1, t_2\rangle$ and we are done. In the second case, the value complexity of $\langle t_1\{v/s\}, t_2\{v/s\}\rangle$, and hence that of the head redex, is either at most the value complexity of $\langle t_1, t_2\rangle$, and we are done, or exactly the value complexity of t, which is smaller than the complexity of the injection redex $\iota_i(t)[x_1.w_1, x_2.w_2]$ occurring in u, which is what we wanted to show. The case in which $\langle t_1, t_2\rangle\, \sigma\{v/s\} = v\, (\sigma_i\{v/s\}) \dots (\sigma_n\{v/s\})$ is impossible due to the form of $s = \iota_i(t)[x_1.w_1, x_2.w_2]$.

- $\iota_i(t')\,\sigma\,\{v/s\}$, where $\sigma = \sigma_1 \dots \sigma_n$ is any case-free stack. Obviously σ must be empty since it is case-free. By induction hypothesis, $(*)$ holds for $t'\{v/s\}$. If $\iota_i(t') = \iota_i(t'\{v/s\})$, all the redexes and channel occurrences that we have to check are in $t'\{v/s\}$ and thus the thesis holds. Indeed, the case in which $\iota_i(t') = v$ is impossible due to the form of $s = \iota_i(t)[x_1.w_1, x_2.w_2]$.

- $x\,\sigma\{v/s\}$, where x is any simply typed variable and $\sigma = \sigma_1 \dots \sigma_n$ is any case-free stack. By induction hypothesis, $(*)$ holds for $\sigma_i\{v/s\}$ for $1 \leq i \leq n$. If $x\,\sigma\{v/s\} = x\,(\sigma\{v/s\})$, all its redexes and channel occurrences are in $\sigma\{v/s\}$, thus the thesis holds. The case in which $x\,\sigma\{v/s\} = v\,(\sigma_i\{v/s\}) \dots (\sigma_n\{v/s\})$ is impossible due to the form of $s = \iota_i(t)[x_1.w_1, x_2.w_2]$.

- $v_0[z_1.v_1, z_2.v_2]\sigma\{v/s\}$, where σ 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

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

  By Lemma 6.2.5 the value complexity of $w_i[t/x_i]$ is either at most the value complexity of w_i or the value complexity of t. In the first case, the value complexity of $w_i[t/x_i]$ is at most the value complexity of w_i which is at most the value complexity of $\iota_i(t)[x_1.w_1, x_2.w_2]$. Thus, by Lemma 6.2.5 the value complexities of $v_0\{v/s\}$, $v_1\{v/s\}$, $v_2\{v/s\}$ are at most the value complexity respectively of v_0, v_1, v_2. Hence, the complexity of the possible case distinction permutation redex

$$(v_0[z_1.v_1\{v/s\}, z_2.v_2\{v/s\}])\sigma_1\{v/s\}$$

  is at most the complexity of $v_0[z_1.v_1, z_2.v_2]\sigma_1$ and we are done. Moreover, the possible injection or case distinction permutation redex

$$v_0\{v/s\}[z_1.v_1\{v/s\}, z_2.v_2\{v/s\}]$$

  satisfies the thesis. In the second case, the value complexities of $v_0\{v/s\}$, $v_1\{v/s\}$, $v_2\{v/s\}$ are at most the value complexities of v_0, v_1, v_2 respectively or exactly the value complexity of t. Hence the complexity of the possible case distinction permutation redex $(v_0[z_1.v_1\{v/s\}, z_2.v_2\{v/s\}])\sigma_1\{v/s\}$ is either at most the value complexity of v_1, v_2, and we are done, or exactly the value complexity of t, which by Proposition 6.2.1 is at most the complexity of the type of t, thus is smaller than the complexity of the injection redex $\iota_i(t)[x_1.w_1, x_2.w_2]$ occurring in u. Moreover, the possible injection or case distinction permutation redex

$$v_0\{v/s\}[z_1.v_1\{v/s\}, z_2.v_2\{v/s\}]$$

  has complexity equal to the value complexity of v_0 or the value complexity of t, and we are done again.

If $v_0[z_1.v_1, z_2.v_2]\sigma\{v/s\} = v\,(\sigma_i\{v/s\})\ldots(\sigma_n\{v/s\})$, then there could be a new intuitionistic redex, when $v = \lambda y\, q$, $v = \langle q_1, q_2\rangle$, $v = \iota_i(q)$ or $v = q_0[y_1.q_1, y_2.q_2]$. If the value complexity of $v = w_i[t/x_i]$ is at most the value complexity of w_i, then the complexity of $w_i[t/x_i](\sigma_i\{v/s\})$ is equal to the complexity of the permutation redex $(\iota_i(t)[x_1.w_1, x_2.w_2])\sigma_i$. If the value complexity of $v = w_i[t/x_i]$ is the value complexity of t, by Proposition 6.2.1 the complexity of $w_i[t/x_i](\sigma_i\{v/s\})$ is at most the complexity of the type of t, thus is smaller than the complexity of the injection redex $\iota_i(t)[x_1.w_1, x_2.w_2]$ occurring in u and we are done.

- $a\,t'\,\sigma\{v/s\}$, 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 $\sigma_i\{v/s\}$ for $1 \le i \le n$. Since $s \neq a\,t'\,\sigma_1\ldots\sigma_j$, the case $a\,t'\,\sigma\{v/s\} = v\,(\sigma_i\{v/s\})\ldots(\sigma_n\{v/s\})$ is impossible.

  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 $a\,(t'\{v/s\})$. By Lemma 6.2.5, the value complexity of $w_i[t/x_i]$ 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 $w_i[t/x_i]$ is at most the value complexity of w_i which is at most the value complexity of $\iota_i(t)[x_1.w_1, x_2.w_2]$. Thus, by Lemma 6.2.5, 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 6.2.1 is at most the complexity of the type of t, thus smaller than the complexity of the injection redex $\iota_i(t)[x_1.w_1, x_2.w_2]$ occurring in u, which is what we wanted to show.

- ${}_a(t_1 \parallel \ldots \parallel t_m)\{v/s\}$. By induction hypothesis, $(*)$ holds for $t_i\{v/s\}$ for $1 \le i \le m$. The only redex in ${}_a(t_1 \parallel \ldots \parallel t_m)\{v/s\}$ and not in some $t_i\{v/s\}$ can be ${}_a(t_1\{v/s\} \parallel \ldots \parallel t_m\{v/s\})\{v/s\}$ 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.

□

We analyze now the complexity of the redexes generated by the permutation of a case distinction.

Lemma 6.2.8 (In the case)**.** *Let u be a term in parallel form. Then for any redex r_1 of Group 1 in $u\{t[x_1.v_1\xi, x_2.v_2\xi]/t[x_1.v_1, x_2.v_2]\xi\}$, there is a redex in u with greater or equal complexity than r_1; for any redex r_2 of Group 2 in $u\{t[x_1.v_1\xi, x_2.v_2\xi]/t[x_1.v_1, x_2.v_2]\xi\}$, there is a redex of Group 2 in u with greater or equal complexity than r_2.*

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

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

$u\{t[x_1.v_1\xi, x_2.v_2\xi]/t[x_1.v_1, x_2.v_2]\xi\}$ there is a redex of Group 2 in u with greater or equal complexity than r_2. Moreover, for any channel occurrence in $u\{v/s\}$ with complexity θ', there is in u an occurrence of the same channel with complexity greater or equal than θ'.

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

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

  $$(\lambda x\, t')\, \sigma\{v/s\} = t[x_1.v_1\xi, x_2.v_2\xi]\, (\sigma_i\{v/s\}) \dots (\sigma_n\{v/s\})$$

  then $t[x_1.v_1\xi, x_2.v_2\xi](\sigma_i\{v/s\})$ can be a new permutation redex. If ξ is case free, this redex has complexity 0 by Lemma 6.2.4 and we are done. If ξ is not case free, $t[x_1.v_1\xi, x_2.v_2\xi](\sigma_i\{v/s\})$ has the same complexity as the permutation $(t[x_1.v_1, x_2.v_2]\xi)\sigma_i$.

- $\langle t_1, t_2\rangle\, \sigma\, \{v/s\}$, where $\sigma = \sigma_1 \dots \sigma_n$ 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

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

  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 $t[x_1.v_1\xi, x_2.v_2\xi]$ is equal to the value complexity of $t[x_1.v_1, x_2.v_2]\xi$. By Lemma 6.2.5, the value complexity of $t_i\{v/s\}$ is at most that of t_i, and we are done. If $\langle t_1, t_2\rangle\, \sigma\{v/s\} = v\, (\sigma_i\{v/s\}) \dots (\sigma_n\{v/s\})$, then
  $v\, (\sigma_i\{v/s\}) = t[x_1.v_1\xi, x_2.v_2\xi](\sigma_i\{v/s\})$ can be a new permutation redex. If ξ is case free, this redex has complexity 0 by Lemma 6.2.4 and we are done. Otherwise,

  $$t[x_1.v_1\xi, x_2.v_2\xi](\sigma_i\{v/s\})$$

  has the same complexity as $(t[x_1.v_1, x_2.v_2]\xi)\sigma_i$.

- $\iota_i(t')\, \sigma\, \{v/s\}$, where $\sigma = \sigma_1 \dots \sigma_n$ is any case-free stack. Obviously, σ must be empty since it is case-free. By induction hypothesis, $(*)$ holds for $t'\{v/s\}$. If $\iota_i(t') = \iota_i(t'\{v/s\})$, all the redexes and channel occurrences that we have to check are in $\sigma\{v/s\}$ and thus the thesis holds.

- $w_0[z_1.w_1, z_2.w_2]\sigma\{v/s\}$, where σ is any case-free stack. By induction hypothesis, $(*)$ holds for $w_0\{v/s\}$, $v_1\{v/s\}$, $v_2\{v/s\}$ and $\sigma_i\{v/s\}$ 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 $t[x_1.v_1\xi, x_2.v_2\xi]$ is equal to the value complexity of $t[x_1.v_1, x_2.v_2]\xi$, by Lemma 6.2.5 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. The complexity of the possible case distinction permutation redex $(w_0\{v/s\}[z_1.w_1\{v/s\}, z_2.w_2\{v/s\}])\sigma_1\{v/s\}$ is thus at most the value complexity of w_1, w_2 respectively, and hence the complexity of the case permutation redex $(w_0[z_1.w_1, z_2.w_2])\sigma_1$. Moreover, the possible injection or case permutation redex

$$w_0\{v/s\}[z_1.w_1\{v/s\}, z_2.w_2\{v/s\}]$$

has complexity equal to the value complexity of w_0 and we are done.

If $w_0[z_1.w_1, z_2.w_2]\sigma\{v/s\} = v\,(\sigma_i\{v/s\})\dots(\sigma_n\{v/s\})$, then there could be a new case distinction permutation redex because $v = t[x_1.v_1\xi, x_2.v_2\xi]$. If ξ is case free, by Lemma 6.2.4, this redex has complexity 0 and we are done; if not, it has the same complexity as $t[x_1.v_1, x_2.v_2]\xi\sigma_1$ and we are done again.

- $x\,\sigma\{v/s\}$, where x is any simply typed variable and $\sigma = \sigma_1 \dots \sigma_n$ 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. If

  $$x\,\sigma\{v/s\} = t[x_1.v_1\xi, x_2.v_2\xi]\,(\sigma_i\{v/s\})\dots(\sigma_n\{v/s\})$$

  then $v\,(\sigma_i\{v/s\})$ can be a new permutation redex. If ξ is case free, this redex has complexity 0 and we are done. Otherwise, $t[x_1.v_1\xi, x_2.v_2\xi](\sigma_i\{v/s\})$ has the same complexity as $t[x_1.v_1, x_2.v_2]\xi\sigma_i$.

- $a\,t\,\sigma\{v/s\}$, where a is a channel variable, t a term and $\sigma = \sigma_1 \dots \sigma_n$ 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 $t[x_1.v_1\xi, x_2.v_2\xi]$ is equal to the value complexity of $t[x_1.v_1, x_2.v_2]\xi$, by Lemma 6.2.5 the value complexity of $t\{v/s\}$ is at most that of t, and we are done.

  If $a\,t\,\sigma\{v/s\} = a\,t\,[x_1.v_1\xi, x_2.v_2\xi]\,(\sigma_i\{v/s\})\dots(\sigma_n\{v/s\})$ then $v\,(\sigma_i\{v/s\})$ can be a new permutation redex. If ξ is case free, this redex has complexity 0 and we are done. Otherwise, $t[x_1.v_1\xi, x_2.v_2\xi](\sigma_i\{v/s\})$ has the same complexity as $(t[x_1.v_1, x_2.v_2]\xi)\sigma_i$.

- ${}_a(t_1 \parallel \dots \parallel t_m)\{v/s\}$. By induction hypothesis, $(*)$ holds for $t_i\{v/s\}$ where $1 \leq i \leq m$. The only redex in ${}_a(t_1 \parallel \dots \parallel t_m)\{v/s\}$ and not in some $t_i\{v/s\}$ can be ${}_a(t_1\{v/s\} \parallel \dots \parallel t_m\{v/s\})$ 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.

□

The following result is meant to guarantee that there cannot be any loop involving intuitionistic redexes and communication redexes of non-decreasing complexity. Intuitively, when Group 1 redexes generate new redexes, the latter have smaller complexity than the former; when Group 2 redexes generate new redexes, the latter does not have greater complexity than the former. Hence, during the communication phase the complexity does not increase and during the intuitionistic phase the complexity strictly decreases.

Proposition 6.2.9 (Decrease!)**.** *Let t be a term in parallel form, r be one of its redexes of complexity τ, and t' be the term that we obtain from t by reducing r.*

1. *If r is a redex of the Group 1, then the complexity of each redex in t' is not greater than the complexity of a redex of the same group occurring in t; or not greater than the complexity of a case distinction permutation redex occurring in t; or smaller than τ.*

2. *If r is a redex of the Group 2 and not an activation redex, then the complexity of any redex in t' is either equal to the complexity of a redex of the same group occurring in t or not greater than τ.*

Proof.

1) Suppose that $r = (\lambda x^A\, s)\, v$, that $s : B$ and let q be a redex in t' whose complexity is different from the complexity of any redex of the same group and any case distinction permutation occurring in t. We apply Lemma 6.2.6 to the term $s[v/x^A]$. From such lemma we can infer that if q occurs in $s[v/x^A]$, since (i) and (ii) do not apply, it has the same value complexity as v, which by Proposition 6.2.1 is at most the complexity of A, and hence strictly smaller than the complexity of $A \to B$ and than the complexity τ of r. Assume hence that q does not occur in $s[v/x^A]$. Since $s[v/x^A] : B$, by applying Lemma 6.2.6 to the term $t' = t\{s[v/x^A]/(\lambda x^A\, s)\, v\}$ we know that either q has the same complexity as the value complexity of $s[v/x^A]$ – which by Proposition 6.2.1 is at most the complexity of B – or q 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 6.2.5 is either at most the complexity of A or at most the complexity of aw.

Suppose that $r = \iota_i(s)[x^A.u_1, y^B.u_2]$. By applying Lemma 6.2.7 to

$$t' = t\{u_i[s/x_i^{A_i}] \,/\, \iota_i(s)[x_1^{A_1}.u_1, x_2^{A_2}.u_2]\}$$

we are done.

2) Suppose that $r = \langle v_0, v_1\rangle\pi_i$ for $i \in \{0, 1\}$, that $\langle v_0, v_1\rangle : A_0 \wedge A_1$ and let q be a redex in t' having complexity greater than that of any redex of the same group in t. The the assumption just made implies that the term q cannot occur in v_i, but it must have been created by the reduction of r. Moreover, by Proposition 6.2.1, the value complexity of v_i cannot be greater than the complexity of A_i. By applying Lemma 6.2.6 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 6.2.6 do not apply. Such complexity is at most the complexity τ of r. Thus we are done.

Suppose that $r = s[x^A.u, y^B.v]\xi$ is a case distinction permutation redex. By applying Lemma 6.2.8 we are done.

If t' is obtained by performing a permutation of a parallelism operator, then obviously the thesis holds.

If t' is obtained by a simplification reduction of the form ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} u_{j_1} \parallel \ldots \parallel u_{j_n}$, for $1 \leq j_1 < \ldots < j_n \leq m$, then there is nothing to prove: all redexes occurring in t' also occur in t.

Suppose now that

$$r = {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

$t_i = \langle u_1^i, \ldots, u_{p_i}^i \rangle$. Then by cross reduction, r reduces to ${}_b(s_1 \parallel \ldots \parallel s_m)$ where if $G_i \neq \bot$:

$$s_i \;=\; {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

and if $G_i = \bot$:

$$s_i \;=\; {}_a(\mathcal{C}_1[a^{G_1 \to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

in which $F_j = G_i$. 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 $\mathcal{C}_i[t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}]$ or $\mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle]$. By applying Lemma 6.2.6 to them, we obtain that the complexity of q is the value complexity of $t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}$ or $b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle$, which, by several applications of Lemma 6.2.5, 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 hence that $q = {}_c(p_1 \parallel \ldots \parallel p_z)$ is a communication redex. Every channel occurrence of c in the terms $\mathcal{C}_i[t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}]$ or $\mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle]$ is of the form $cw\{t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j} / a\, t_i\}$ or $cw\{b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle / a\, t_i\}$ where cw is a channel occurrence in t. By Lemma 6.2.5, each of these occurrences has either at most the value complexity of cw or has at most the value complexity of r, which is again a contradiction. □

The following result is meant to guarantee that a loop between communication redexes and activation redexes is impossible: no new activation is generated after a cross reduction if there is none to start with.

Lemma 6.2.10 (Freeze!)**.** *Suppose that s is a term in parallel form that does not contain projection, case distinction permutation or activation redexes. Let ${}_a(q_1 \parallel \ldots \parallel q_m)$ be some redex in s of complexity τ. If s' is obtained from s by performing first a cross reduction on ${}_a(q_1 \parallel \ldots \parallel q_m)$ and then reducing all projection and case distinction permutation redexes, then s' contains no activation redexes.*

Proof. Let ${}_a(q_1 \parallel \ldots \parallel q_m) = {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1\sigma_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m\sigma_m])$ where σ_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', or in other terms $s' = s\{t'/t\}$. Then after performing the cross reduction and reducing all the intuitionistic redexes, t reduces to ${}_b(s_1 \parallel \ldots \parallel s_m)$ where if $G_i \neq \bot$:

$$s_i \;=\; {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[t'_j] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

and if $G_i = \bot$:

$$s_i \;=\; {}_a(\mathcal{C}_1[a^{G_1 \to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

in which $F_j = G_i$ and t'_j are the terms obtained reducing all projection and case distinction permutation redexes in $t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}$.

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 $b\langle \boldsymbol{y}_i \rangle$, t' is not an activation redex. Now we consider the channel occurrences in any term s_i. We first show that there are no activable channels in t'_j which are bound in s'. Since for any stack θ of projections $b\langle \boldsymbol{y}_i \rangle\theta$ has value complexity 0, for any subterm w of t_j we can apply repeatedly Lemma 6.2.5 to $w^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}$ and obtain that the value complexity of $w^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}$ is exactly the value complexity of w. This implies that there is no activable channel in $t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}$ because there was none in t_j. The following general statement immediately implies that there is no activable channel in t'_j which is bound in s' either.

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

We prove ($*$) by induction on the size of r and proceed by cases according to its shape.

- If $r = \lambda x w$, $r = \iota_i(w)$, $r = w\mathsf{efq}_P$, $r = x$ or $r = aw$, for a channel a, then $r' = r\theta$ and the thesis holds.
- If $r = \langle v_0, v_1 \rangle$ the only redex that can occur in $r\theta$ is a projection redex, when $\theta = \pi_i\rho$. Hence, $r\theta \mapsto_{\mathrm{L}} v_i\rho \mapsto^*_{\mathrm{L}} r'$. Since v_i cannot be a channel because otherwise $\langle v_0, v_1 \rangle$ would not be a legal term, we can apply the induction hypothesis on v_i and ρ. Hence, there are no activable channels in r' which are bound in s'.
- If $r = t[x.v_0, y.v_1]$, then $r\theta \mapsto^*_{\mathrm{L}} t[x.v_0\theta, y.v_1\theta] \mapsto^*_{\mathrm{L}} t[x.v'_0, y.v'_1] = r'$. By induction hypothesis applied to v_0 and θ and to v_1 and θ, there are no activable channels in v'_0 and v'_1 and we are done.
- If $r = p\nu\xi$, with ξ case free, then $r' = p\nu\xi$ and the thesis holds.

Now, let c be any non-active channel bound in s' occurring in $\mathcal{C}_i[t'_j]$ or $\mathcal{C}_i[b^{B_i \to \perp}\langle \boldsymbol{y}_i \rangle]$ but not in u': any of its occurrences is of the form $c\langle p_1, \ldots, p_l\{u'/av\rho\}, \ldots, p_n\rangle$, where each p_l is not a pair. We want to show that the value complexity of $p_l\{u'/av\rho\}$ is exactly the value complexity of p_l. Indeed, $p_l = r\,\nu$ where ν is a case-free stack. If r is of the form λxw, $\langle q_1, q_2\rangle$, $\iota_i(w)$, x, dw, with $d \neq a$, then the value complexity of $p_l[u'/av\rho]$ is the same as that of p_l; note, indeed, that if $r = \langle q_1, q_1\rangle$, then ν is not empty because $r\,\nu$ is assumed not to be a pair. If $r = v_0[x_1.v_1, x_2.v_2]$, then ν is empty, otherwise s would contain a permutation redex, so $c\langle p_1, \ldots, p_l, \ldots, p_n\rangle$ is activable, and there is an activation redex in s, which is contrary to our assumptions. The case $r = av$ and $\nu = \rho\rho'$ is also impossible, otherwise $c\langle p_1, \ldots, p_l, \ldots, p_n\rangle$ would be activable, and we are done. □

Definition 6.2.9. The height of a term t in parallel form is

- 0 if t is a simply typed λ-term
- $1 + \max(m, n)$ if $t = u \parallel_a v$ and the heights of u and v are m and n respectively.

We show that the communication phase of our reduction strategy is finite.

Lemma 6.2.11 (Communicate!)**.** *Let t be any term in parallel form that does not contain projection, case distinction permutation, or activation redexes. Assume moreover that all redexes in t have complexity at most τ. Then t reduces to a term containing no redexes, except Group 1 redexes of complexity at most τ.*

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

- n is the number of subterms ${}_a(u_1 \parallel \ldots \parallel u_m)$ of t such that ${}_a(u_1 \parallel \ldots \parallel u_m)$ 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 ${}_a(u_1 \parallel \ldots \parallel u_m)$ in t containing m occurrences of a.

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 $f(i) < f'(i)$.

If $h(j) > 0$, for some $j \geq 2$, then there is at least an active session ${}_a(u_1 \parallel \ldots \parallel u_m)$ in t that does not contain any active session and such that the height of ${}_a(u_1 \parallel \ldots \parallel u_m)$ is j. Hence $u_i = {}_b(s_1 \parallel \ldots \parallel s_q)$ for some $1 \leq i \leq i \leq m$. We obtain t' by applying inside t the permutation ${}_a(u_1 \parallel \ldots \parallel {}_b(s_1 \parallel \ldots \parallel s_q) \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_b({}_a(u_1 \parallel \ldots \parallel s_1 \ldots \parallel u_m) \ldots \parallel {}_a(u_1 \parallel \ldots \parallel s_q \ldots \parallel u_m))$. We claim that the term t' thus obtained has complexity has complexity (n, h', g'), with $h' < h$. Indeed, ${}_a(u_1 \parallel \ldots \parallel u_m)$ 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 hence $h' < h$. 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 ${}_a(u_1 \parallel \ldots \parallel u_m)$ in t such that the height of ${}_a(u_1 \parallel \ldots \parallel u_m)$ is 1; if there is none, we are done. We reason by cases on the distribution of the occurrences of a. Either (i) some u_i for $1 \leq i \leq m$ 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 simplification reduction ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} u_{j_1} \parallel \ldots \parallel u_{j_p}$. If there is an active session ${}_b(s_1 \parallel \ldots \parallel s_q)$ in t such that ${}_a(u_1 \parallel \ldots \parallel u_m)$ is the only active session contained in some s_i for $1 \leq i \leq n$, then the term t' has complexity $(n-1, h', g')$, because ${}_b(s_1 \parallel \ldots \parallel s_q)$ is an active session which is not uppermost in t, but is uppermost in t'; if not, we claim that the term t' has complexity (n, h, g') where $g' < g$. Indeed, first, the number of active sessions which are not uppermost does not change. Second, the height of all other uppermost active sessions does not change. Third, $g'(j) = g(j) - 1$ and, for any $i \neq j$, $g'(i) = g(i)$ because, obviously, no channel belonging to any uppermost active session different from ${}_a(u_1 \parallel \ldots \parallel u_m)$ occurs in u. Since the reduction ${}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} u_{j_1} \parallel \ldots \parallel u_{j_p}$ does not introduce any new intuitionistic or activation redex, we can apply the induction hypothesis on t' and obtain the thesis.

Suppose now that (ii) is the case and that all u_i for $1 \leq i \leq m$ together contain j occurrences of a. Then ${}_a(u_1 \parallel \ldots \parallel u_m)$ is of the form ${}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$ where a is active, $\mathcal{C}_j[a^{F_j \to G_j}\, t_j]$ for $1 \leq j \leq m$ are simply typed λ-terms; $a^{F_j \to G_j}$ is rightmost in each of them. Then we can apply the cross reduction ${}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m]) \mapsto_{\mathrm{L}} {}_b(s_1 \parallel \ldots \parallel s_m)$ in which b is fresh and for $1 \leq i \leq m$, we define, if $G_i \neq \bot$:

$$s_i \;=\; {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

and if $G_i = \bot$:

$$s_i \;=\; {}_a(\mathcal{C}_1[a^{G_1 \to G_1}\, t_1] \ldots \parallel \mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle] \parallel \ldots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$$

where $F_j = G_i$; $\boldsymbol{y}_z$ for $1 \leq z \leq m$ is the sequence of the free variables of t_z bound in $\mathcal{C}_z[a^{F_z \to G_z}\, t_z]$; B_z for $1 \leq z \leq m$ is the type of $\langle \boldsymbol{y}_z \rangle$. By Lemma 6.2.10, after performing all projections and case permutation reductions in all $\mathcal{C}_i[t_j^{\,b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}]$ and $\mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle]$ for $1 \leq i \leq m$ we obtain a term t' that contains no activation redexes; moreover, by point 2. of Proposition 6.2.9, t' contains only redexes having complexity at most τ.

We claim that the term t' thus obtained has complexity $\langle n, h, g' \rangle$ where $g' < g$. Indeed, the value n does not change because all newly introduced occurrences of b are not active. The new active sessions $s_i = {}_a(\mathcal{C}_1[a^{F_1 \to G_1}\, t_1] \dots \parallel \mathcal{C}_i[t_j^{b^{B_i \to B_j}\langle \boldsymbol{y}_i \rangle / \boldsymbol{y}_j}] \parallel \dots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$ or $s_i = {}_a(\mathcal{C}_1[a^{G_1 \to G_1}\, t_1] \dots \parallel \mathcal{C}_i[b^{B_i \to \bot}\langle \boldsymbol{y}_i \rangle] \parallel \dots \parallel \mathcal{C}_m[a^{F_m \to G_m}\, t_m])$ 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 ${}_a(u_1 \parallel \dots \parallel u_m)$ 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 to prove that the normalization procedure, applied to a proof term in parallel form, terminates and yields a proof term in normal form.

Proposition 6.2.12 (Normalize!)**.** *Let $t : A$ be any term in parallel form. Then $t \mapsto^*_{\mathrm{L}} t'$, where t' is in parallel normal form.*

Proof. Let τ be the maximum among the complexity of the redexes in t. We prove the statement by induction on τ. Starting from t, we reduce all intuitionistic redexes and obtain a term t_1 that, by Proposition 6.2.9, does not contain redexes of complexity greater than τ. By Lemma 6.2.3, $t_1 \mapsto^*_{\mathrm{L}} t_2$ where t_2 does not contain any redex, except cross reduction redexes of complexity at most τ. By Lemma 6.2.11, $t_2 \mapsto^*_{\mathrm{L}} t_3$ where t_3 contains only Group 1 redexes of complexity at most τ. Suppose $t_3 \mapsto^*_{\mathrm{L}} t_4$ by reducing all Group 1 redexes, starting from t_3. By Proposition 6.2.9, every Group 1 redex generated in the process has complexity at most τ, thus every Group 2 redex which is generated has complexity smaller than τ , thus t_4 can only contain redexes with complexity smaller than τ. By induction hypothesis $t_4 \mapsto^*_{\mathrm{L}} t'$, with t' in parallel normal form. □

The normalization of all $\lambda_{\parallel \mathrm{L}}$ proof terms easily follows.

Theorem 6.2.13 (Normalization theorem)**.** *Suppose that $t : A$ is a $\lambda_{\parallel \mathrm{L}}$ proof term. Then $t \mapsto^*_{\mathrm{L}} t' : A$, where t' is a normal parallel form.*

Proof. By Proposition 6.2.2 and 6.2.12. □

Remark 6.4. The normalization procedure presented here would work for any type system containing rules extracted from axioms of the form

$$\bigvee_{i=1,\dots,k} \Big(\bigwedge_{j=1,\dots,n_k} A_i^j \Big) \to B_i$$

where A_i^j and B_i, for $i = 1, \dots, k$ is a propositional variable or $\top$ or $\bot$, with the additional restriction that all A_j^i's are distinct and each $A_j^i \neq \top$ is equal to some B_l. Such class is a

proper generalization of the class considered here, but the normalization procedure and the normalization proof can be easily adapted to it without any significant change. In spite of this, the notation of typing rules and reductions would greatly suffer in clarity from such generalization. Hence we only present the simplified version of the procedure and the relative proof.

6.3 The subformula property

We show that normal $\lambda_{\|\mathrm{L}}$-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 $\lambda_{\|\mathrm{L}}$ is meaningful from the logical perspective and produces analytic proofs.

We first show that every normal form $\lambda_{\|\mathrm{L}}$-term is in parallel form.

Proposition 6.3.1 (Parallel normal form property)**.** *If $t \in \mathrm{NF}_{\mathrm{L}}$ is a $\lambda_{\|\mathrm{L}}$-term, then it is in parallel 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 ${}_a(u_1 \parallel \ldots \parallel u_m)$, otherwise one could apply the permutation
$$t = \lambda x^A\, {}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_a(\lambda x^A\, u_1 \parallel \ldots \parallel \lambda x^A.u_m)$$
and t would not be in normal form. Hence, by induction hypothesis v must be a simply typed λ-term.
- $t = \langle v_1, v_2\rangle$. Since t is normal, neither v_1 nor v_2 can be of the form ${}_a(u_1 \parallel \ldots \parallel u_m)$, otherwise one could apply one of the permutations
$$\langle {}_a(u_1 \parallel \ldots \parallel u_m),\, w\rangle \mapsto_{\mathrm{L}} {}_a(\langle u_1, w\rangle \parallel \ldots \parallel \langle u_m, w\rangle)$$
$$\langle w,\, {}_a(u_1 \parallel \ldots \parallel u_m)\rangle \mapsto_{\mathrm{L}} {}_a(\langle w, u_1\rangle \parallel \ldots \parallel \langle w, u_m\rangle)$$
and t would not be in normal form. Hence, by induction hypothesis v_1 and v_2 must be simply typed λ-terms.
- $t = v_1\, v_2$. Since t is normal, neither v_1 nor v_2 can be of the form ${}_a(u_1 \parallel \ldots \parallel u_m)$, otherwise one could apply one of the permutations
$${}_a(u_1 \parallel \ldots \parallel u_m)\, w \mapsto_{\mathrm{L}} {}_a(u_1 w \parallel \ldots \parallel u_m w)$$
$$w\, {}_a(u_1 \parallel \ldots \parallel u_m) \mapsto_{\mathrm{L}} {}_a(w u_1 \parallel \ldots \parallel w u_m)$$
and t would not be in normal form. Hence, by induction hypothesis v_1 and v_2 must be simply typed λ-terms.

- $t = v\,\mathsf{efq}_P$. Since t is normal, v cannot be of the form ${}_a(u_1 \parallel \ldots \parallel u_m)$, otherwise one could apply the permutation

$$ {}_a(u_1 \parallel \ldots \parallel u_m)\,\mathsf{efq}_P \mapsto_{\mathrm{L}} {}_a(u_1\,\mathsf{efq}_P \parallel \ldots \parallel u_m\,\mathsf{efq}_P) $$

  and t would not be in normal form. Hence, by induction hypothesis $u_1, \ldots, u_m$ must be simply typed λ-terms.

- $t = u\,\pi_i$. Since t is normal, v can be of the form ${}_a(u_1 \parallel \ldots \parallel u_m)$, otherwise one could apply the permutation

$$ {}_a(u_1 \parallel \ldots \parallel u_m)\,\pi_i \mapsto_{\mathrm{L}} {}_a(u_1\pi_i \parallel \ldots \parallel u_m\pi_i) $$

  and t would not be in normal form. Hence, by induction hypothesis u must be a simply typed λ-term, which is the thesis.

- $t = {}_a(u_1 \parallel \ldots \parallel u_m)$. By induction hypothesis the thesis holds for u_i where $1 \leq i \leq m$ and hence trivially for t.

□

We now show that the subformula property holds for normal $\lambda_{\parallel \mathrm{L}}$-terms.

Theorem 6.3.2 (Subformula property)**.** *Suppose*

$$ x_1^{A_1}, \ldots, x_n^{A_n}, a_1^{D_1}, \ldots, a_m^{D_m} \vdash t : A \quad \textit{and} \quad t \in \mathrm{NF}_{\mathrm{L}}. \quad \textit{Then}: $$

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

Proof. We proceed by structural induction on t and reason by cases, according to the form of t.

- $t = \langle u, v \rangle : F \wedge G$. Since $t \in \mathrm{NF}_{\mathrm{L}}$, by Proposition 6.3.1 it is in parallel form, thus is a simply typed λ-term. Hence no communication variable can be bound inside t, thus 1. trivially holds. By induction hypothesis, 2. holds for $u : F$ and $v : G$. Hence, the type of any subterm of u is either a subformula or a conjunction of subformulae of $A_1, \ldots, A_n$, of F and of proper subformulae of $D_1, \ldots, D_m$ and any subterm of v is either a subformula or a conjunction of subformulae of some $A_1, \ldots, A_n$, of G and of proper subformulae of $D_1, \ldots, D_m$. Moreover, any subformula of F and G must be a subformula of the type $F \wedge G$ of t. Hence the type of any subterm of $\langle u, v \rangle$ is either a subformula or a conjunction of subformulae of $A_1, \ldots, A_n, F \wedge G$ or a proper subformula of $D_1, \ldots, D_m$ and the statement holds for t as well.

- $t = \lambda x^F\, u : F \to G$. Since $t \in \mathrm{NF_L}$, by Proposition 6.3.1 it is in parallel form, thus is a simply typed λ-term. Hence no communication variable can be bound inside t, thus 1. trivially holds. By induction hypothesis, 2. holds for $u : G$. Hence the type of any subterm of u is either a subformula or a conjunction of subformulae of some $A_1, \ldots, A_n, F$, of G and of proper subformulae of $D_1, \ldots, D_m$. Since the type F of x is a subformula of $F \to G$, the type of any subterm of $\lambda x^F\, u$ is either a subformula or a conjunction of subformulae of $A_1, \ldots, A_n, F \to G$ or a proper subformula of $D_1, \ldots, D_m$ and the statement holds for t as well.

- $t = \iota_i(u) : F \vee G$ for $i \in \{0, 1\}$. Without loss of generality assume that $i = 1$ and $u : G$. Since $t \in \mathrm{NF_L}$, by Proposition 6.3.1 it is in parallel form, thus is a simply typed λ-term. Hence no communication variable can be bound inside t, thus 1. trivially holds. By induction hypothesis, 2. holds for $u : F$. Hence, the type of any subterm of u is either a subformula or a conjunction of subformulae of some $A_1, \ldots, A_n$, of F or proper subformulae of $D_1, \ldots, D_m$. Moreover, any subformula of G must be a subformula of the type $F \vee G$ of t. Hence the type of any subterm of $\iota_i(u)$ is either a subformula or a conjunction of subformulae of $A_1, \ldots, A_n, F \vee G$ or a proper subformula of $D_1, \ldots, D_m$ and the statement holds for t as well.

- $t = x^{A_i}\, \sigma : A$ for some A_i among $A_1, \ldots, A_n$ and stack σ. Since $t \in \mathrm{NF_L}$, it is in parallel form, thus is a simply typed λ-term and no communication variable can be bound inside t. By induction hypothesis, for any element $\sigma_j : S_j$ of σ, the type of any subterm of σ_j is either a subformula or a conjunction of subformulae of some $A_1, \ldots, A_n$, of the type S_j of σ_j and of proper subformulae of $D_1, \ldots, D_m$.

  If σ is case-free, then every S_j is a subformula of A_i, or of A, when $\sigma = \sigma' \mathsf{efq}_A$. Hence, the type of any subterm of $x^{A_i}\, \sigma$ is either a subformula or a conjunction of subformulae of $A_1, \ldots, A_n, A$ or of proper subformulae of $D_1, \ldots, D_m$ and the statement holds for t as well.

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

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

- $t = {}_b(u_1 \parallel \ldots \parallel u_k) : A$ and $b^{G_i \to H_i}$ 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 $u_1 : A, \ldots, u_k : A$. We first show that it cannot be the case that

(∗) all prime factors of $G_1, H_1, \ldots, G_k, H_k$ are subformulae of $A_1, \ldots, A_n, A$ or proper subformulae 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 formulae B, C of any bound communication variable $a^{B \to C}$ of t. Let P be any prime factor of T or B or C. By induction hypothesis applied to $u_1, \ldots, u_n$, we obtain that P is either subformula or conjunction of subformulae of $A_1, \ldots, A_n, A$ and of proper subformulae 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 $a^{F_1 \to F_2}$ 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 $D_1, \ldots, D_m$ and (ii) for every other bound channel variable $c^{S_1 \to S_2}$ of t, if some prime factor Q' of S_1 or S_2 is neither a subformula of $A_1, \ldots, A_n, A$ nor a proper subformula of $D_1, \ldots, D_m$, then Q' is complex at most as Q. If Q is a subformula of F_1 we say that $a^{F_1 \to F_2}$ 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 $c^{F_1 \to F_2} w$ 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 t_1 be any subterm of t such that t_1 contains at least a maximally violating channel and all maximally violating channel of t that are free in t_1 are maximally violating in the input. Then there is a simply typed subterm s of t_1 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 t_1. channels.

If $n = 0$, it is enough to pick $s = t_1$.

If $n > 0$, let $t_1 = {}_d(v_1 \parallel \ldots \parallel v_n)$ and assume $d^{E_i \to F_i}$ occurs in v_i. If no $d^{E_i \to F_i}$ maximally violates the subformula property, we obtain the thesis by applying the induction hypothesis to any v_i. Assume hence that some $d^{E_i \to F_i}$ maximally violates the subformula property due to Q. Then there is some $d^{E_j \to F_j}$ such that Q is a prime factor of E_i or E_j. By induction hypothesis applied to u_i or u_j, 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 $c^{F_1 \to F_2} w$ 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 the term $c^{F_1 \to F_2} w$. Since Q is a prime factor of F_1, it is either an atom P or a formula of the form $Q' \to Q''$ or of the form $Q' \vee Q''$. Let $w = \langle w_1, \ldots, w_j \rangle$, where each w_i is not a pair, and let k be such that that Q occurs in the type of w_k.

We start by ruling out the case that $w_k = \lambda y\, s$ or $w_k = \iota_i(s)$ for $i \in \{0, 1\}$, otherwise it would be possible to perform an activation reduction or a cross reduction to some subterm ${}_c(u'_1 \parallel \ldots \parallel u'_{m'})$, which must exist since c is bound.

Suppose now, by contradiction, that $w_k = x^T\, \sigma$ where σ is a stack. It cannot be the case that $\sigma = \sigma'[y^{E_1}.v_1, z^{E_2}.v_2]$ or that $\sigma = \sigma' \mathsf{efq}_P$, because otherwise we could apply an activation reduction or a cross reduction. Hence σ is case-free and does not contain efq_P. Moreover, x^T cannot be a free variable of t, because then T would be equal to some A_i for $1 \leq i \leq n$, and Q would be a subformula of A_i, which contradicts the assumptions. Suppose hence that x^T is a bound intuitionistic variable of t, such that t has a subterm $\lambda x^T s : T \to Y$ or, without loss of generality, $s[x^T.v_1, z^E.v_2]$, with $s : T \vee Y$ for some formula Y. By induction hypothesis $T \to Y$ and $T \vee Y$ are subformulae of $A_1, \ldots, A_n, A$ or proper subformulae of $D_1, \ldots, D_m, G \to H$. But $T \to Y$ and $T \vee Y$ contain Q as a proper subformula and $c^{F_1 \to F_2}\, w$ violates maximally the subformula due to Q. Hence $T \to Y$ and $T \vee Y$ are neither subformulae of $A_1, \ldots, A_n, A$ nor proper subformulae of $D_1, \ldots, D_m$ and thus must be proper subformulae of $G \to H$. Since $c^{F_1 \to F_2}\, w$ violates the subformula property maximally due to Q, $T \to Y$ and $T \vee Y$ must be at most as complex as Q, which is a contradiction. Suppose now that x^T is a bound channel variable, thus $w_k = a^T\, r\, \sigma$, where a^T is a bound communication variable of t, with $T = T_1 \to T_2$. Since $c^{F_1 \to F_2}\, w$ is rightmost, $a \neq c$. Moreover, Q is a subformula of a prime factor of T_2, whereas $a^{T_1 \to T_2}$ occurs in w, which is impossible by choice of c. This contradicts the assumption that the term is normal and ends the proof.

□

CHAPTER 7

Conclusion and future work

In this dissertation we defined Curry–Howard correspondences for individual logics and families of intermediate logics, including well-known systems such as classical and Gödel–Dummett logic. These correspondences give rise to typed concurrent and parallel λ-calculi which are more expressive than simply typed λ-calculus and exhibit interesting computational properties. This confirms Avron's intuitions on the computational content of the intermediate logics which are naturally formalized as hypersequent calculi [Avr91].

The presented results can be considered satisfactorily conclusive in this respect. Nevertheless, they open new research lines, some of which concern promising ways of extending the work.

The first future work direction that we discuss is related to the proof-theoretical techniques employed in Chapter 3 to bridge hypersequents and natural deduction through systems of rules. The idea at the origin of systems of rules is to use non-local conditions inside derivations in order to avoid the addition of explicit structural elements for increasing the expressive power of sequent calculus. This idea and its effectiveness is neither limited to the intermediate logics studied here, nor to intermediate logics as a whole. The original version of systems of rules was introduced, for instance, to define calculi based on the frame semantics of modal logics; and we can formalize these logics by systems of rules in a purely syntactic way [Pav18]. Non-local conditions can also be used as means to define analytic calculi for the sub-structural logics that can be formalized as hypersequent calculi. In each of these cases, a smart use of such conditions could not only lead to the definition of analytic calculi with interesting properties, but also to the definition of simple natural deduction calculi. The success in defining suitable analytic natural deduction calculi, in turn, could lead to new computational interpretations of the corresponding logics.

From the point of view of concurrency theory, it would be interesting to show a formal result relating the concurrent λ-calculi presented in this dissertation and π-calculus. An encoding between a typed fragment of π-calculus and one of the $\lambda_{\|\mathrm{L}}$-calculi would indeed

enable us to pinpoint the exact relationship between the two kinds of formalisms. The task seems achievable, for example, by defining more liberal versions of the cross reductions of λ_{G}. A promising way to define more sophisticated calculi for modeling concurrent systems is to use linear logic instead of intuitionistic logic as the base logic. In particular, a linear version of the natural deduction calculi presented in this dissertation should correspond to concurrent λ-calculi in which we can control the number of communications between processes.

A first step towards the definition of a parallel programming language based on intermediate logics would be to implement the reduction system of $\lambda_{\|}$, as a standalone interpreter or using the constructs of an existing programming language. The calculus $\lambda_{\|}$ could also constitute a solid base for endeavors using logic for the specification and synthesis of parallel functional programs. While there are several type systems for concurrent calculi, none of them is based on a sufficiently expressive logical language, as first-order arithmetic or second-order logic would be. On the other hand, extending propositional Curry–Howard correspondences to higher-order logics is not only a possible but also a natural development of the present work. Nevertheless, there is no obvious way to do it because the applicability conditions of the usual quantifier rules interfere with the transformation of proofs into *parallel normal forms* – see Propositions 4.1.8 and 6.2.2. This normal form, in turn, is essential to the normalization procedures used in this dissertation. Thus, new techniques are probably required to define normalizing first-order or second order calculi based on the ideas employed here.

Finally, a problem of concurrency theory which has remained open for several decades is the definition of a formalism providing a general foundation for concurrent program evaluation. For sequential computation, such a foundation is provided by λ-calculus. According to Milner, the introduction of CCS itself – the forefather of process calculi and, in particular, of π-calculus – was precisely due to the unsuccessful attempts to define satisfactory concurrent extensions of λ-calculus [Mil84]. While the definition of such an untyped concurrent λ-calculus lies outside the scope of this dissertation, the set of reductions presented for $\lambda_{\|\mathrm{L}}$ might shed some light on possible approaches to the solution of this problem.

Appendix

Here is the unabridged reduction of the three-process $\lambda_{\|}$ implementation of the Floyd–Warshall algorithm discussed in Section 5.6.3.

$$
\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\Big(\big(f(a(f(a(f(aI_1(0)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (aI_1(0))\pi_0\big)\\
&\quad\parallel \underline{\big(f(a(f(a(f(a(f(aI_2(0)))))))) \parallel (a((a((a((a((aI_2(0))\pi_1))\pi_1))\pi_1))\pi_1))\pi_0}\\
&\quad\quad\underline{\parallel (a(f(a(f(aI_2(0)))))\pi_0 \parallel (a((aI_2(0))\pi_1))\pi_0\big)}\\
&\quad\parallel \big(f(a(f(a(f(a(f(aI_3(0)))))))) \parallel (a((a((a((aI_3(0))\pi_1))\pi_1))\pi_1))\pi_0\\
&\quad\quad\parallel (a(f(a(f(a(f(aI_3(0)))))))\pi_0 \parallel (a((a((aI_3(0))\pi_1))\pi_1))\pi_0\big)\big)
\end{aligned}
$$

$$
\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\Big(\big(f(a(f(a(f(aI_1(0)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(I_1(0)))\pi_0\big)\\
&\quad\parallel \big(f(a(f(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))) \parallel (a((a((a((a(\langle I_2(0), I_1(0)\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_0 \parallel\\
&\quad\quad(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))\pi_0 \parallel (a(\langle I_2(0), I_1(0)\rangle\pi_1))\pi_0\big)\\
&\quad\parallel \underline{\big(f(a(f(a(f(a(f(aI_3(0)))))))) \parallel (a((a((a((a(\langle 0, I_3(0)\rangle\pi_1))\pi_1))\pi_1))\pi_1))\pi_0}\\
&\quad\quad\parallel \underline{(a(f(a(f(a(f(aI_3(0)))))))\pi_0 \parallel (a((a((a(\langle 0, I_3(0)\rangle\pi_1))\pi_1))\pi_1))\pi_0\big)\big)}
\end{aligned}
$$

$$
\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\Big(\big(f(a(f(a(f(aI_1(0)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(I_1(0)))\big)\pi_0\\
&\quad\parallel \big(f(a(f(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(f(a(f\langle I_2(0), I_1(0)\rangle)))))\pi_0 \parallel (aI_1(0))\pi_0\big)\\
&\quad\parallel \underline{\big(f(a(f(a(f(a(f(aI_3(0)))))))) \parallel (a((a((a((aI_3(0))\pi_1))\pi_1))\pi_1))\pi_0}\\
&\quad\quad\parallel \underline{(a(f(a(f(a(f(aI_3(0)))))))\pi_0 \parallel (a((a((aI_3(0))\pi_1))\pi_1))\pi_0\big)\big)}
\end{aligned}
$$

$$
\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\Big(\big(\underline{f(a(f(a(f(aI_1(0)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(I_1(0)))\pi_0}\big)\\
&\quad\parallel \big(f(a(f(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(f(a(f\langle I_2(0), I_1(0)\rangle)))))\pi_0 \parallel (aI_1(0))\pi_0\big)\\
&\quad\parallel \big(f(a(f(a(f(a(f\langle I_3(0), I_1(0)\rangle)))))) \parallel (a((a((a(\langle I_3(0), I_1(0)\rangle\pi_1))\pi_1))\pi_1))\pi_0\\
&\quad\quad\parallel (a(f(a(f(a(f\langle I_3(0), I_1(0)\rangle)))))\pi_0 \parallel (a((a(\langle I_3(0), I_1(0)\rangle\pi_1))\pi_1))\pi_0\big)\big)
\end{aligned}
$$

$$
\begin{aligned}
&\mapsto^{*}_{\mathtt{p}\,a}\Big(\big(\underline{f(a(f(a(f(aI_1(0)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(I_1(0)))\pi_0}\big)\\
&\quad\parallel \big(f(a(f(a(f(a(f\langle I_2(0), I_1(0)\rangle)))))) \parallel (a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0 \parallel (a(f(a(f\langle I_2(0), I_1(0)\rangle)))))\pi_0 \parallel (aI_1(0))\pi_0\big)\\
&\quad\parallel \big(f(a(f(a(f(a(f\langle I_3(0), I_1(0)\rangle)))))) \parallel (a((a((a(I_1(0)))\pi_1))\pi_1))\pi_0\\
&\quad\quad\parallel (a(f(a(f(a(f\langle I_3(0), I_1(0)\rangle)))))\pi_0 \parallel (a((aI_1(0))\pi_1))\pi_0\big)\big)
\end{aligned}
$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(a(f(a(f\langle I_1(0),I_1(0)\rangle)))))\parallel(a((a((a(\langle I_1(0),I_1(0)\rangle\pi_1))\pi_1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(\underline{f(a(f(a(f(a(f\langle I_2(0),I_1(0)\rangle))))))\parallel(a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0\parallel(a(f(a(f\langle I_2(0),I_1(0)\rangle)))))\pi_0\parallel(aI_1(0))\pi_0}\big)$$
$$\parallel\big(f(a(f(a(f(a(f\langle I_3(0),I_1(0)\rangle))))))\parallel(a((a((a(I_1(0)))\pi_1))\pi_1))\pi_0$$
$$\parallel(a(f(a(f(a(f\langle I_3(0),I_1(0)\rangle)))))\pi_0\parallel(a((aI_1(0))\pi_1))\pi_0\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(a(f(aI_1(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(f(a(f(a(f(a(f(aI_2(1))))))\parallel(a((a((a((aI_1(0))\pi_1))\pi_1))\pi_1))\pi_0\parallel(a(f(aI_2(1)))\parallel aI_1(0))\pi_0\big)$$
$$\parallel\big(f(a(f(a(f(a(f(aI_3(1))))))\parallel(a((a((a(I_1(0)))\pi_1))\pi_1))\pi_0\parallel(a(f(a(f(aI_3(1))))))\pi_0\parallel(a((aI_1(0))\pi_1))\pi_0\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(a(f(aI_1(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel(\langle I_1(0),I_1(0)\rangle)\pi_0\big)$$
$$\parallel\big(f(a(f(aI_2(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel(a(I_2(1)))\pi_0\parallel(\langle I_1(0),I_1(0)\rangle)\pi_0\big)$$
$$\parallel\big(\underline{f(a(f(a(f(a(f(aI_3(1))))))\parallel(a((a((a(I_1(0)))\pi_1))\pi_1))\pi_0\parallel(a(f(a(f(aI_3(1))))))\pi_0\parallel(a((aI_1(0))\pi_1))\pi_0}\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(\underline{f(a(f(aI_1(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0}\big)$$
$$\parallel\big(f(a(f(aI_2(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel(a\,I_2(1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(f(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\parallel(a((a(\langle I_3(1),I_2(1)\rangle\pi_1)\pi_1))\pi_0$$
$$\parallel(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\pi_0\parallel(a(\langle I_1(0),I_2(1)\rangle\pi_1))\pi_0\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(\underline{f(a(f(aI_1(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0}\big)$$
$$\parallel\big(f(a(f(aI_2(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel(aI_2(1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(f(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel(a(f(aI(f\langle I_3(1),I_2(1)\rangle)))))\pi_0\parallel(aI_2(1))\pi_0\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(a(f\langle I_1(1),I_2(1)\rangle)))\parallel(a((a(\langle I_1(0),I_2(1)\rangle\pi_1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(\underline{f(a(f(aI_2(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel(aI_2(1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0}\big)$$
$$\parallel\big(f(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\pi_0\parallel(aI_2(1))\pi_0\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(a(f\langle I_1(1),I_2(1)\rangle)))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(\underline{f(a(f(aI_2(1))))\parallel(a((a((aI_1(0))\pi_1))\pi_1))\pi_0\parallel(aI_2(1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0}\big)$$
$$\parallel\big(f(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\pi_0\parallel(aI_2(1))\pi_0\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(a(f\langle I_1(1),I_2(1)\rangle)))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(f(a(f\langle I_2(1),I_2(1)\rangle)))\parallel(a((a(\langle I_1(0),I_2(1)\rangle\pi_1))\pi_1))\pi_0\parallel\langle I_2(1),I_2(1)\rangle\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(\underline{f(a(f(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel(a(f(a(f\langle I_3(1),I_2(1)\rangle)))))\pi_0\parallel(aI_2(1))\pi_0}\big)\big)$$

$$\mapsto^*_{\mathrm{p}\,a}\big(\big(f(aI_1(2))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(f(aI_2(2))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel\langle I_2(1),I_2(1)\rangle\pi_0\parallel\langle I_1(0),I_1(0)\rangle\pi_0\big)$$
$$\parallel\big(\underline{f(a(f(aI_3(2))))\parallel(a((aI_2(1))\pi_1))\pi_0\parallel(a(f(aI_3(2))))\pi_0\parallel(aI_2(1))\pi_0}\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(\underline{f(aI_1(2)) \parallel (a((aI_2(1))\pi_1))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)$$
$$\parallel \big(f(aI_2(2)) \parallel (a((aI_2(1))\pi_1))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(f(a(f\langle I_3(2), I_2(1)\rangle)) \parallel (a(\langle I_2(1), I_2(1)\rangle\pi_1))\pi_0 \parallel (a(f\langle I_3(2), I_2(1)\rangle))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(\underline{f(aI_1(2)) \parallel (a((aI_2(1))\pi_1))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)$$
$$\parallel \big(f(aI_2(2)) \parallel (a((aI_2(1))\pi_1))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(f(aI_3(2)) \parallel (aI_2(1))\pi_0 \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(f\langle I_1(2), I_3(2)\rangle \parallel (a(\langle I_2(1), I_3(2)\rangle\pi_1))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(\underline{f(aI_2(2)) \parallel (a((aI_2(1))\pi_1))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)$$
$$\parallel \big(f(aI_3(2)) \parallel (aI_2(1))\pi_0 \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(f\langle I_1(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(\underline{f(aI_2(2)) \parallel (a((aI_2(1))\pi_1))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)$$
$$\parallel \big(f(aI_3(2)) \parallel (aI_2(1))\pi_0 \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(f\langle I_1(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(f\langle I_2(2), I_3(2)\rangle \parallel (a(\langle I_2(1), I_3(2)\rangle\pi_1))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(\underline{f(aI_3(2)) \parallel (aI_2(1))\pi_0 \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0}\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(f\langle I_1(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(f\langle I_2(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(\underline{f(aI_3(2)) \parallel (aI_2(1))\pi_0 \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0}\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(\underline{f\langle I_1(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)$$
$$\parallel \big(f\langle I_2(2), I_3(2)\rangle \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(f\langle I_3(2), I_3(2)\rangle \parallel \langle I_2(1), I_3(2)\rangle\pi_0 \parallel \langle I_3(2), I_3(2)\rangle\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big)$$

$$\mapsto^*_{\mathtt{p}\,a}\big(\big(\underline{I_1(3) \parallel (aI_3(2))\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0}\big)$$
$$\parallel \big(I_2(3) \parallel (aI_3(2))\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0 \parallel \langle I_1(0), I_1(0)\rangle\pi_0\big)$$
$$\parallel \big(I_3(3) \parallel \langle I_2(1), I_3(2)\rangle\pi_0 \parallel \langle I_3(2), I_3(2)\rangle\pi_0 \parallel \langle I_2(1), I_2(1)\rangle\pi_0\big)\big) \quad \mapsto^*_{\mathtt{p}} I_1(3) \parallel I_2(3) \parallel I_3(3)$$

Index

Bibliography

[ACG17] Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. Gödel logic: from natural deduction to parallel computation. In *LICS 2017*, pages 1–12, 2017.

[ACG18] Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. Classical proofs as parallel programs. In *GandALF 2018*, pages 43–57, 2018.

[ACG19a] Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. A concurrent computational interpretation of intermediate logics. Submitted for publication, 2019.

[ACG19b] Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. A parallel λ-calculus for graph-based communication. Submitted for publication, 2019.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[Asc16] Federico Aschieri. On natural deduction for herbrand constructive logics i: Curry–howard correspondence for dummett's logic lc. *Log. Methods Comput. Sci.*, 12(13):1–31, 2016.

[Avr87] Arnon Avron. A constructive analysis of RM. *The Journal of Symbolic Logic*, 52(4):939–951, 1987.

[Avr91] Arnon Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Annals of Mathematics and Artificial Intelligence*, 4(3):225–248, 1991.

[AZ16] Federico Aschieri and Margherita Zorzi. On natural deduction in classical first-order logic: Curry–howard correspondence, strong normalization and herbrand's theorem. *Theoretical Computer Science*, 625:125–146, 2016.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus, its Syntax and Semantics*. Amsterdam: North-Holland, 1984.

[BCF00] Matthias Baaz, Agata Ciabattoni, and Christian Fermüller. A natural deduction system for intuitionistic fuzzy logic. In *Lectures on Soft Computing and Fuzzy Logic*, pages 1–18. Physica-Verlag, 2000.

[Bou89] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In *TAPSOFT 1998*, pages 149–161, 1989.

[BP15] Arnold Beckmann and Norbert Preining. Hyper natural deduction. In *LICS 2015*, pages 547–558. IEEE Computer Society, 2015.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[CG16] Agata Ciabattoni and Francesco A. Genco. Embedding formalisms: hypersequents and two-level systems of rules. In *Advances in Modal Logic*, volume 11, pages 197–216. College Publications, 2016.

[CG18] Agata Ciabattoni and Francesco A. Genco. Hypersequents and systems of rules: Embeddings and applications. *ACM Transactions on Computational Logic (TOCL)*, 19(2):11:1–11:27, 2018.

[CGT08] Agata Ciabattoni, Nikolaos Galatos, and Kazushige Terui. From axioms to analytic rules in nonclassical logics. In *LICS 2008*, pages 229–240. IEEE Computer Society, 2008.

[CMS18] Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. Choreographies, logically. *Distributed Computing*, 31(1):51–67, 2018.

[CP10a] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR 2010*, pages 222–236, 2010.

[CP10b] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR 2010*, pages 222–236, 2010.

[CZ97] Alexander Chagrov and Michael Zakharyaschev. *Modal Logic*, volume 35 of *Oxford Logic Guides*. Oxford University Press, 1997.

[dG95] Philippe de Groote. A simple calculus of exception handling. In *TLCA 1995*, pages 201–215, 1995.

[DK00] Vincent Danos and Jean-Louis Krivine. Disjunctive tautologies as synchronisation schemes. *CSL 2000*, 1862:292–301, 2000.

[Dum59] Michael Dummett. A propositional calculus with denumerable matrix. *The Journal of Symbolic Logic*, 24(2):97–106, 1959.

[EBPJ11] Jeff Epstein, Andrew P. Black, and Simon L. Peyton Jones. Towards haskell in the cloud. In *ACM Haskell Symposium 2011*, pages 118–129, 2011.

[FFKD86] Matthias Felleisen, Daniel P Friedman, Eugene E Kohlbecker, and Bruce F Duba. Reasoning with continuations. In *LICS 1986*, pages 131–141, 1986.

[FPV98] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. In *IEEE Transactions on Software Engineering*, volume 24, pages 342–361, 1998.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, 39(1):176–210, 1935.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[Göd32] Kurt Gödel. Zum intuitionistischen Aussagenkalkül. *Anzeiger der Akademie der Wissenschaftischen in Wien*, 69:65–66, 1932.

[Gri90] Timothy G. Griffin. A formulae-as-type notion of control. In *POPL 1990*, 1990.

[Hey30] Arend Heyting. Die formale regeln der intuitionistischen logik. *Sitzungsberichte der Preussischen Akademie von Wissenschaften, Physikalisch Methematische Klasse*, pages 158–169, 1930.

[Hir12] Yoichi Hirai. A lambda calculus for gödel–dummett logic capturing waitfreedom. In *FLOPS 2012*, pages 151–165, 2012.

[HL13] Thomas Horstmeyer and Rita Loogen. Graph-based communication in eden. *Higher-order and symbolic computation*, 26(1):3–28, 2013.

[How80] William A. Howard. The formulae-as-types notion of construction. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–491. Academic Press, 1980.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *ECOOP 1991*, pages 133–147, 1991.

[KMP19] Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Better late than never: a fully-abstract semantics for classical processes. In *POPL 2019*, pages 24:1–24:29, 2019.

[Kri90] Jean-Louis Krivine. Lambda-calcul types et modèles. In *Studies in Logic and Foundations of Mathematics*, pages 1–176. Masson, 1990.

[Kri09] Jean-Louis Krivine. Classical realizability. *Interactive models of computation and program behavior, Panoramas et synthèses*, pages 197–229, 2009.

[Lan64] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[LE82a] Edgar G.K. López-Escobar. Implicational logics in natural deduction systems. *The Journal of Symbolic Logic*, 47:184–186, 1982.

[LE82b] Edgar G.K. López-Escobar. Implicational logics in natural deduction systems. *The Journal of Symbolic Logic*, 47(1):184–186, 1982.

[Loo11] Rita Loogen. Eden – parallel functional programming with haskell. In *CEFP 2011*, pages 142–206, 2011.

[LOP05] Rita Loogen, Yolanda Ortega-Mallén, and Ricardo Peña-Marí. Parallel functional programming in eden. *Journal of Functional Programming*, 15(3):431–475, 2005.

[MCHP04] Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *LICS 2004*, pages 286–295, 2004.

[Mil84] Robin Milner. Lectures on a calculus for communicating systems. In *Seminar on Concurrency*, pages 197–219. Springer, 1984.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Neg05] Sara Negri. Proof analysis in modal logic. *Journal of Philosophical Logic*, 34:507–544, 2005.

[Neg16] Sara Negri. Proof analysis beyond geometric theories: from rule systems to systems of rules. *Journal of Logic and Computation*, 27:513–537, 2016.

[Par92] Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR 1992*, pages 190–201, 1992.

[Pav18] Sanja Pavlović. Proof theory for modal logics: Embedding between hypersequent calculi and systems of rules. Master's thesis, TU Wien, 2018.

[Pra71] Dag Prawitz. Ideas and results in proof theory. In *Proceedings of the Second Scandinavian Logic Symposium*, 1971.

[San93] Davide Sangiorgi. *Expressing mobility in process algebras: first-order and higher-order paradigms.* PhD thesis, The University of Edinburgh, 1993.

[SH14] Peter Schroeder-Heister. The calculus of higher-level rules, propositional quantification, and the foundational approach to proof-theoretic harmony. *Studia Logica*, 102(6):1185–1216, 2014.

[SU98] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Elsevier, 1998.

[SW03] Davide Sangiorgi and David Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2003.

[TCP13] Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *ESOP 2013*, pages 350–369, 2013.

[TS96] Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. *SIGPLAN Notices*, 38(9):189–201, 2003.

[Wad12] Philip Wadler. Propositions as sessions. *ICFP 2012*, 24:384–418, 2012.

[Wad15] Philip Wadler. Propositions as types. *Communications of the ACM*, 58(12):75–84, 2015.