École Polytechnique
Thèse de Doctorat
Spécialité Informatique

Nested deduction in logical foundations for computation

Présentée et soutenue publiquement par

Nicolas Guenot

le 10 Avril 2013

devant le jury composé de

Delia	Kesner	Rapporteur
Richard	McKinley	
Dale	Miller	
Luca	Roversi	Rapporteur
Lutz	Strassburger	Directeur de thèse
Benjamin	Werner	

Nested Deduction in Logical Foundations for Computation

Nicolas Guenot

April 8, 2013

Acknowledgements

A PhD thesis is always a long story... well at least it was for me, and of course it involves a lot of people providing help and support. For the thesis itself, I have to thank Lutz Straßburger — accepting to be my supervisor, and for the freedom I got in my research. Also, I owe very much to Dale Miller, for being so helpful on scientific questions as well as more down-to-earth questions. He is responsible for the nice research environment one can find in the Parsifal team, from the lively discussions to the many opportunities to meet researchers and work with different people. It was a great pleasure to work with other members of the team, and I am particularly thankful to Kaustuv Chaudhuri for our collaboration and for the many interesting discussions.

I would like to thank Delia Kesner for accepting to review thoroughly this thesis, and Roy Dyckhoff even if this finally could not go all the way... Also, Luca Roversi for his kindness taking part as a reviewer. I am honored that Richard McKinley and Benjamin Werner accepted to be on the jury, along with Dale and Lutz. I must thank Stéphane Lengrand, Catuscia Palamidessi and Christine Ferret for their help in the process of organising the defense.

Just getting to the point of actually starting a PhD thesis is a long road, which was rather bumpy in my case. For helping me make the right choices at crossroads, I am very grateful to Christine Paulin, who made me end up in Lyon — wise choice with fortunate consequences — to Daniel Hirschkoff for his brilliant teaching and the motivation to pursue that comes along. Of course my studies would not have been so interesting without the good patronage of Tom Hirschowitz, or the famous » *td-men* « from Brice Goglin to Emmanuel Jeandel and Jeremie Detrey.

Slowly moving from studies to research, I should thank many persons met on the way: Kai Brünnler, Alessio Guglielmi, Paola Bruscoli, Michel Parigot, and Tom Gundersen, Willem Heijltjes, and also Olivier Delande, Alexis Saurin, Vivek Nigam, and Beniamino Accattoli, Pierre Boudes, and others, for stimulating conversations. A special *thank you very much* to David Baelde for his help with scientific matters as in other situations. Thanks to the many nice people met at LIX, PPS and LIPN, my research and teaching time in Paris was very enjoyable.

Because student life is not only about learning from books, I owe many thanks to the » *gang des lyonnais et assimilés* « starting with Stéphane, Camille, Pierre-Yves, Vinz, Jules, Jade, et Cédric, Pierre, Benoit, Claire, Lucie, Samuel... and others. In Paris, even more people, I shall pay a couple of biers to those I forget: thank you Matthias, Clément, Maxime and Jasmine, Marina, thank you Jonas, Fabien and the others for the welcoming atmosphere of PPS. Many people and places should be properly thanked for making my life nicer. Salut à toi rue Bobillot, Folie en Tête, et rue de Crimée. Vielen Dank, Heidelberg and Leipzig and all of you multitude of flatmates for your warm hospitality. Thank you Céline, Maman, as usual.

Finally, for me not going too insane over the years, *merci* Nina.

Contents

Introduction

Logic was born in a time when mathematicians would build complex theories on the grounds of abstract objects that one can hardly understand without diving into the definitions, theorems and technical proofs of the whole field of research, but it has proved to be a crucial guidance on the way to the separation between what mathematicians can prove, and what they can actually build. Interestingly enough, formal logic was developed by philosophers and mathematicians such as Russell and Whitehead with the explicit purpose of making *everything* formal, leading a new generation, under the influence of logicians like Brouwer, to reject entire parts of mathematics for being based only on formal objects that cannot be constructed. Those who were trying to rebuild mathematics on the grounds of the finitist and constructivist methods made crucial steps towards the radically new understanding of a foundamental concept, that was rarely expressed in mathematics and never described as such in any formal study: the notion of computation.

Some decades after the construction of the first computing machines that could execute various tasks described by the means of programs, the intimate connection between logic and computation was made clear with the new bridge established by Curry and Howard, and others, between the foundational theory of mathematical constructivism, the deductive system of intuitionistic logic, and a most remarkable model of computation, the λ-calculus. The correspondence is rather simple: proofs are programs, and programs are proofs. This gives to the formulas of intuitionistic logic the status of types, an abstraction that allows to classify programs and give proofs of their good properties. The slogan for introducing types and their logical background into the practice of programming sounds convincing, since its states that » *well-typed programs never go wrong* «, and this eventually lead to the creation of new paradigms and languages, derived from the λ-calculus and structured by logic, allowing for concise and elegant programs.

The logical approach to computation tends to reject the untyped programs and to consider proofs and logical systems as completely describing the interesting part of computation, with a nice consequence for logic: the necessity of inventing new ways of programming to face new challenges such as external faults, interactivity or concurrency was an efficient incentive and a source of inspiration for logicians to refine their understanding of the laws of logic. In a few decades, the traditional Curry-Howard correspondence was extended and refined, to discover the relation between computation and classical logic as well as new logics stemming from the study of models of programming, such as linear logic.

The current state of the logical approach to computation is interesting, because the rapidly developing study of programming methods goes far beyond what logic can explain, but there is always a benefit in trying to capture these new mechanisms in a principled, logical approach. Apart from the proofs-as-programs methodology, the concept of logic programming has been introduced to improve the expressivity of languages and take advantage of the consistent framework offered by logic. On the side of proof theory, recent developments have established new possibilities of manipulating and reasoning about proofs, and therefore programs. There are many new concepts to explore, which will surely bring interesting insights on both logic and computation.

★★★

The structural approach to proof theory emphasises the importance of designing deductive proof systems based on inference rules that respect certain criterions. In particular, a subtle equilibrium is necessary to ensure that even if it contains rules like the *cut* rule introduced by Gentzen in the sequent calculus, to formalise the use of lemmas, a system is complete with respect to a given logic in its *analytic* form — that is, rules such as cut are admissible in the system, only rules for decomposing formulas are necessary. This result of *normalisation*, or *cut elimination*, is essential to the two main approaches to computation followed and developped by logicians, functional programming and logic programming. The procedures required to build a normal proof from any given proof has been thoroughly investigated in the setting of standard, *shallow* formalisms such as natural deduction and the sequent calculus, where formulas are decomposed from the outside.

However, structural proof theory is an active field of research, where new ways of representing proofs have been proposed, in particular since the inception of the graphical system of proof-nets [Gir96] for linear logic. The so-called *deep inference* methodology [Gug07] is another recent development which aims at overcoming a particular limitation of the sequent calculus: its *» sequential «* access to the contents of formulas, which is problematic for example when dealing with a certain form of non-commutativity [Tiu06b]. In this setting, inference rules can be applied directly inside formulas, so that the whole deduction process takes the form of a logically sound *rewriting* of formulas. The formalism of the *calculus of structures* is the most representative example of this methodology, and it has been used to define proof systems for various logics, such as classical logic [Brü03], linear logic [Str03a] and variants of these. However, in this setting, the shape of proofs and the large number of possible configurations in the composition of the inference rule instances makes it difficult to apply the traditional methods of structural proof theory.

In such a *nested* setting, where inference rules apply deep inside formulas that are themselves modified by some other rule instances, the study of foundamental normal forms of proofs, such as analytic proofs — that would be cut-free proofs in the sequent calculus — or the uniform [MNPS91] and focused proofs [And92], has not yet lead to a systematic approach for defining normalisation procedures or proving the completeness of normal fragments. In particular, several techniques have been used for proving cut elimination for systems in the calculus of structures,

by substitutive methods [Brü06a], or by reducing proofs to a certain shape with the so-called *splitting* lemma [GS11b], or through graphical representations of the flow of atoms in proofs [GGS11]. The kind of normal forms used in a proof search perspective have not been much studied, although proof search in the calculus of structures has been investigated, from an implementation viewpoint [Kah06] and in its applications [Bru02]. In any case, the tools developped to study these aspects of proofs in the deep inference setting are radically different from the standard ones used in shallow formalisms, and this most probably the principal reason why computational interpretations for nested systems have been neglected: outside of the standard frameworks of normalisation and its connexion to the λ-calculus, and of focusing and its use in the definition of logic programming languages, it is more difficult to study the computational contents of logics, since this would require to develop a matching theory of computation.

This seems to lead to the conclusion that it is a highly difficult task to describe the computational contents of logics through their presentation in the calculus of structures. But if standard computational devices can be invoked as soon as we use the standard tools for normal forms in natural deduction and the sequent calculus, a solution to the problem is to develop the equivalent of the standard normalisation and focusing techniques in the deep inference setting:

> • *the thesis of this work is that it is possible to transfer the technology used for cut elimination and focusing in shallow formalisms to the setting of nested proof systems, and that this leads to computational interpretations identical to or refining the interpretations given for the standard systems in natural deduction and sequent calculi* •

This claim will be supported here by two developments, to transfer the standard techniques for normalisation, through permutations of rule instances, and focusing, to proof systems in the calculus of structures, and in nested sequents [Brü10]. The normalisation procedure defined is then used as the basis for type systems, relating it to the standard interpretation of proofs in natural deduction and in the sequent calculus. On the side of logic programming, the focused system obtained is closely related to the standard focusing system for linear logic in the sequent calculus.

The original grounds for the Curry-Howard correspondence, relating the proofs of intuitionistic natural deduction to the pure λ-calculus, or variants using explicit substitutions, are well-suited for this transfer of technology: it is a minimal example of this kind of computational interpretation, and relate proofs to a well-understood framework. However, intuitionistic logic has not often been investigated, and the systems proposed in the calculus of structures are not designed in a way that makes such an interpretation easy. One of them emphasises the design of local inference rules [Tiu06a], but it does not offer an internal procedure for cut elimination. The only system designed to provide some computational interpretation [BM08] lacks a terminating normalisation procedure, so that we need to define new systems, for adapting the standard techniques based on permutations. All the systems we will describe here can be thought of as generalisations of well-known natural deduction or sequent calculus systems, with the purpose of facilitating permutation. The main

contributions on the side of logic there are the procedures for cut elimination and normalisation in intuitionistic systems. Moreover, cut elimination is generalised to a symmetric normalisation procedure, that should lead to a refinement of the usual methodology of explicit substitutions. Finally, the proof systems defined induce an adaptation of the standard typing methodology which allow to show how standard λ-calculi with explicit substitutions can be used as a computational interpretation of proofs in this setting. Moreover, specific features of the nested proof systems are exploited to introduce new operators in the syntax of λ-calculi, allowing a complex control over the operational behaviour of terms, although the resulting calculi have poor properties in the untyped setting, so that a further study would be required to fully understand the non-standard computational contents of proofs in the calculus of structures.

On the side of logic programming, the rich syntax for proofs in the calculus of structures is used to refine the interpretation that can be made of the proof search process in terms of computation. The main contribution here is the transposition of the *focusing* technique [And92], initially developped for linear logic in the sequent calculus, to the calculus of structures, based on the standard system [Str03a], where it appears in a decomposed form that exposes the crucial role of polarities. It is here surprising that, beyond the close correspondence that can be established with the usual focused sequent calculus, the proof of completeness of the focused calculus of structures can be done through some rather straightforward proof transformation, based on permutations of rule instances — to eliminate the one inference rule that breaks focusing, much like cut can be eliminated to produce analytic proofs. In this system, the *» focusing phases «* from the shallow setting are decomposed into slices that describe the interaction of a unique negative formula with a complete positive synthetic connective. Such a decomposition should lead to a refined interpretation of focused proof search in terms of computation.

Moreover, a new correspondence is described between proof search in a simple proof system for intuitionistic logic in the calculus of structures and reduction in a λ-calculus with explicit substitutions, through the interpretation of intuitionistic formulas as λ-terms. Notice that this goes beyond the scope of adapting standard techniques to the nested setting, since this correspondence cannot be established in shallow formalisms, but this might lead to a better understanding of the relation between the two major approaches to logical aspects of computation, even in the standard setting.

Synopsis. This thesis is divided into four parts. The first one is introductory for all the other parts, and the third one depends one the second part — the last one is relatively independent. Each chapter is made as self-contained as possible. The main contributions on the side of logic are the cut elimination and normalisation proofs of Chapter 3 and Chapter 4, and the description of the focused system along with its completeness proof in Chapter 7. The other chapters of the last two parts describe the particular use of deep inference systems on the side of computation.

Part 1 — Preliminaries

The goal of this part is to recall known definitions and results in structural proof theory and on λ-calculi with explicit substitutions, and establish the notations and basic proof techniques used in other parts. It also describes non-standard material, or material that is rarely developped in the literature.

Chapter 1. Standard and Nested Proof Theory

The standard systems for intuitionistic logic, in both settings of natural deduction and the sequent calculus, are recalled. The general concept of *» deep inference «* as a logical methodology is exposed and presented under its two principal forms, nested sequents and the calculus of structures, both illustrated with examples from classical logic. Some basic tools for rule instances permutations and the analysis of the structure of proofs are described.

Chapter 2. Logical Foundations for Computation

The standard correspondence between proofs in the intuitionistic systems and λ-terms, with or without explicit substitution, is recalled. The notions of typing and type system are described and the correspondence is established between variants of the natural deduction system for intuitionistic logic and various λ-calculi with explicit substitutions, as well as between sequent calculi variants and an alternative presentation of λ-calculi based on a sequentialised `let/in` application syntax. The sequent calculus for linear logic and its properties are recalled, and its use in logic programming, through the use of the focusing technique, is presented.

Part 2 — Intuitionistic Logic in Deep Inference

This part is concerned with the developement of a proof theory for intuitionistic logic in a deep inference setting, it contains the definition of various systems, and the main results of cut elimination, which are used is the rest of the thesis.

Chapter 3. Intuitionistic Logic in Nested Sequents

A family of proof systems for intuitionistic logic in nested sequents is presented, and compared to the standard sequent calculus systems. The admissibility of the cut rule is proved, through a purely syntactic procedure for eliminated cut instances from a proof. A generalised symmetric system is introduced, and the corresponding normalisation procedure, a generalisation of cut elimination, is described.

Chapter 4. Intuitionistic Logic in the Calculus of Structures

The previous nested sequent intuitionistic system is transposed into the calculus of structures and simplified. A different system, based on a natural deduction style, is described. A detour elimination procedure is defined to compute normal forms, yielding a simpler system of proof rewritings than the systems in sequent style.

Part 3 — Nested Proofs as Programs

This part describes the main computational interpretation of the proof systems for intuitionistic logic of the previous part, by generalising the notion of type system to fit the generalised shape of proofs in a deep inference system, and suggesting a computational meaning to the characteristic features of the calculus of structures.

Chapter 5. Nested Typing for Explicit Substitutions

The standard definition of type system is extended into a setting where a typing judgement contains other typing judgements. The correspondence between such typing derivations and proofs of intuitionistic logical systems in the deep inference setting yields a connection between these nested proofs and λ-terms with explicit substitutions, with or without the `let/in` syntax. Standard results are proved for this notion of typing, including normalisation of typed term.

Chapter 6. Nested Typing and Extended λ-calculi

The implicit co-contraction induced by contraction in the calculus of structures, and the decomposition of the context splitting of standard formalisms performed by the switch rule are used as guidance in the definition of two λ-calculi extended with new operators. In the first one, a rule based on contraction provides a way of typing an operator for building a superposition of distinct subterms, introducing a general notion of resources possibly sharing a context. In the second one, communication operators inspired from the π-calculus are extracted from the use of the switch in a type system, yielding a λ-calculus where the distribution of explicit substitutions, considered as resources, is explicitly controlled by links between certain subterms, which are considered as parallel threads in a program.

Part 4 — Nested Proof Search as Computation

This part investigates the use of deep inference systems to perform proof search and the connection between nested proof search procedures and other computational devices. Since proof search is directly impacted by the design of proof systems, it is also concerned with the shape of inference rules and the normal forms of proofs.

Chapter 7. Nested Focusing in Linear Logic

A proof system for linear logic in the calculus of structures is defined, and modified to provide a reasonable framework for proof search. Introducing explicit polarity shifts in the syntax of formulas leads to the definition of a focused system, which enjoys a particularly simple, direct proof of completeness. This system implements an incremental version of the standard concept of focusing, which is compared to the standard focused sequent calculus.

Chapter 8. Proof Search as Reduction in the λ-calculus

The intuitionistic calculus of structures in sequent style presented in Chapter 4 is restricted to obtain a subsystem where inference rules are in exact correspondence with the reduction rules of a λ-calculus with explicit substitutions, through some encoding of λ-terms into formulas of this system. A focused variant of this system is defined, which is shown to correspond to a big-step, call-by-name implementation of the λ-calculus.

PART 1

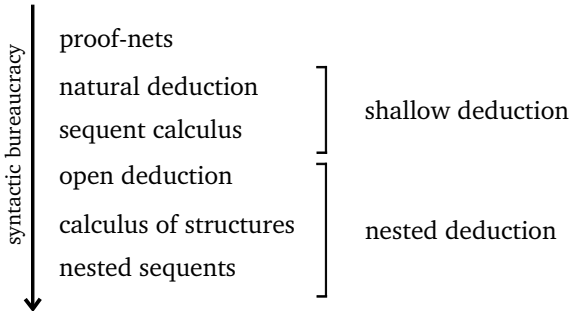
Preliminaries

Chapter 1

Standard and Nested Proof Theory

This chapter is concerned with the basics of the » *structural approach* « to proof theory, and it provides background for the rest of the document, starting with the standard proof theory developped in the formalisms of natural deduction and the sequent calculus. In particular, our study of proof systems and their computational interpretations will have a strong emphasis on *normal forms* that can be obtained by a purely syntactic process of permutations of rule instances. The tools required to study permutations are defined, and the standard proof systems for intuitionistic logic are recalled, as they will form the basis for most of the developments of other chapters — those concerned with typed λ-calculi.

The non-standard contents of this chapter are the descriptions of two formalisms based on the » *deep inference* « methodology, called the calculus of structures and nested sequents, which are the setting in which we attempt to lay the foundations for a logical approach to refined computational models. The syntactic nature of the proof objects using nested deduction, where are not applied only at toplevel as in shallow formalisms, is such that more permutations of rule instances are allowed, yielding more » *bureaucracy* « but also more possibilities for the design of rules.



The basic definitions are given, as well as tools required to handle permutations, which can be more complex in this setting than in shallow formalisms, in both the calculus of structures and nested sequents, and with examples from classical logic.

1 Proofs and Standard Logical Formalisms

In the early times of formal logic, the notion of proof was not providing any object to be manipulated easily, due to the lack of a simple syntax, which would be rich enough to describe the fundamental structure of deductive reasoning. Indeed, the only important structure in deductive logic was for long the *modus ponens* scheme, which is, informally, the following rule:

if ϕ implies ψ and ϕ holds, then ψ holds

where ϕ and ψ are any valid logical propositions. For example, the so-called *Hilbert systems* [BH34] usually have only this rule as an inference rule, and can represent different logics using different sets of axioms. All theorems are then derived from axioms by composition, using this rule. In such a setting, it is difficult to develop a theory of proofs as mathematical objects, and this reflects the main interest in the field at the time, which was to know whether a proposition is provable, rather than knowing if there are different proofs of the same proposition and how they relate.

The situation changed radically when Gentzen established the foundations of *structural proof theory*, by introducing the formalisms of natural deduction and the sequent calculus, in his seminal papers [Gen34, Gen35]. In these formalisms as in other modern logical formalisms, proof objects are part of a well-structured theory, organised in layers, from the level of logical propositions to the level of inference, providing a syntax for the result of a proof construction process. We provide here a description of these layers, as well as meta-level observations on this theory.

1.1 Logical Formulas and Judgements

In the *propositional* setting, we have to assume given an infinite, countable set $\mathscr{A}$ of *atoms*, denoted by small latin letters such as a, b and c. When needed, we will also assume given a bijective function $\overline{\cdot} : \mathscr{A} \mapsto \mathscr{A}$ called *negation*, such that $\overline{a} \neq a$ for any a. Atoms are meant to represent basic logical propositions that can be formed in the language of discourse. In a first-order setting, with quantifiers, this set must be generalised to a set of infinitely many *predicates* of all possible arities, which can be applied to terms, as usually defined in the literature [TS96].

The language of a logic is formed, based on atoms or predicates, by connectives used to compose propositions into complex ones. Connectives can have different arities — although they are usually considered to have either zero, one, two, or an arbitrary and variable number of arguments — and nullary connectives are called *units*, because they are used to represent the units of other connectives.

Definition 1.1. *A* formula *is an expression built from atoms, connectives and units, viewed through its* syntax tree*, where inner nodes are connectives and leaves are atoms or units — and a node formed by some n-ary connective has n child nodes.*

Given a particular logic, its logical language of formulas is usually given by a recursive grammar. Similarly, we can define formulas inductively, accepting atoms and units as valid formulas, and considering connectives as functions from formula tuples to formulas. We denote formulas by capital latin letters such as A, B, C.

Example 1.2. *In classical logic, we have the two connectives* $\wedge$ *and* $\vee$ *which represent conjunction and disjunction respectively, so that we can write the formula* $A \vee (B \wedge C)$*, which should be read as* » either A holds or both B and C hold, or all of them hold « *given some formulas A, B and C.*

In some situations, it might prove useful to consider that the connectives used in a logic have good properties, such as associativity and commutativity. This can be done for example by using a *congruence relation* on formulas and then dealing with the equivalences classes. Indeed, the intended meaning of connectives usually matches these properties, and we can then write, in classical logic, $A \vee B \vee C$ rather than $(A \vee B) \vee C$ or $A \vee (B \vee C)$, or any commutative variant of these. This relates to the use of n-ary connectives, since the formula $\bigvee A_i$ using n-ary disjunction is a way of avoiding the question of associativity.

It is also common to extend the negation function to formulas, using a bijection between formulas, such as $\neg$ in classical logic. It can then interact with negation as defined for atoms, if $\neg a = \overline{a}$ for any a. Moreover, it defines important dualities between two connectives, as between the conjunction and disjunction connectives in classical logic[1], where $\neg(A \vee B) = \neg A \wedge \neg B$ and $\neg(A \wedge B) = \neg A \vee \neg B$.

During the process of validating a given formula with respect to a certain logic, showing it is a *theorem* of this logic, we can keep track of which parts of this formula have been treated and which ones are left to be handled. Parts of the initial formula are then often organised in two groups: subformulas being considered as valid and used as *assumptions*, and other subformulas left to validate, the *goal* of the proof construction process. This is formalised through the following notion.

Definition 1.3. *A* sequent *is a pair* $\Gamma \vdash \Delta$ *where* Γ *and* Δ *are possibly empty multisets of formulas, and* Γ *is called the* antecedent *of the sequent, and* Δ *its* succedent.

We are using multisets here to simplify definitions and notations, but one should notice that plain sets can also be used, and some logics require sequents to use lists, such as non-commutative [AR99] or cyclic logics [Yet90]. A sequent is sometimes called a *judgement*, as it is a statement on the acceptability of a formulas.

1.2 Inference Rules, Proofs and Systems

Now that the level of logical propositions is defined, we can proceed to the level of *inference* — the act of deducing, from a set of premises, that a given conclusion is valid. The rich structure of modern logical formalisms comes from the syntax they provide for the various deduction mechanisms available in a logic. Indeed, we can define a syntax for one inference step following a valid scheme, as shown below.

Definition 1.4. *An* inference rule *is a scheme formed of a conclusion* $\Gamma \vdash \Delta$ *and a set of* $n \geq 0$ *premises* $\Psi_1 \vdash \Sigma_1, \cdots, \Psi_n \vdash \Sigma_n$*, denoted as follows:*

$$\mathsf{r}\ \frac{\Psi_1 \vdash \Sigma_1 \quad \cdots \quad \Psi_n \vdash \Sigma_n}{\Gamma \vdash \Delta}$$

[1] Although it seems at first sight to be a natural notion, as in classical logic, negation in general can raise complex questions on the behaviour of logics and proof systems [Mel10].

An inference rule is given as a scheme, where variables are meant to be replaced with actual formulas and multisets of formulas. A rule having no premise is called an *axiomatic* rule. An *instance* of an inference rule is then any instantiation of the scheme associated to the rule. The meaning of a rule as described in the definition above is that from a set of sequents of the shape described in the rule, and accepted as premises, we can conclude that another sequent of the shape described by the conclusion of the rule is valid in the considered logic.

Example 1.5. *In the sequent calculus for classical logic, we have the following rule:*

$$\wedge_{\mathsf{R}} \frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$$

which means that in order to prove a sequent where some formula of the shape $A \wedge B$ appears, one has to be able to prove this sequent where only A appears, and also this sequent where only B appears.

The point of using such inference rule instances is that they can be composed, in a syntactic way, to represent the causal chain that justifies the validity of a sequent with the validity of other sequents, or axioms. The fundamental object of structural proof theory is then such a composition, which represents a flow of inference steps, a complex deduction. Because of the *branching* structure induced by the possibility of having several premises in a rule, it is represented as a tree. Such a derivation will be denoted by the letter $\mathscr{D}$, possibly with indices.

Definition 1.6. *A* derivation *is a rooted tree where nodes are inference rule instances, such that for any node v with a parent w, there is one premise in w which is equal to the conclusion of v.*

Example 1.7. *In the sequent calculus for classical logic, we can build a derivation of conclusion $A, B \vee D \vdash A \wedge (B \vee C)$ and with one open premise $A, B \vee D \vdash B, C$:*

$$\wedge_{\mathsf{R}} \frac{\mathsf{ax}\,\dfrac{}{A, B \vee D \vdash A} \qquad \vee_{\mathsf{R}} \dfrac{A, B \vee D \vdash B, C}{A, B \vee D \vdash B \vee C}}{A, B \vee D \vdash A \wedge (B \vee C)}$$

Using this notion, we can manipulate objects representing partially completed deductive processes, the validity of conclusions depending on a future verification of the validity of premises. This follows the scheme of inference rules, so that we can actually decide, when needed, to consider new rules formed by composition of other rules, thus hiding a derivation inside the box of one inference step. Such a rule is called *synthetic*, as the one below, which can be formed in classical logic:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad = \qquad \vee_{\mathsf{R}} \frac{\mathsf{weak}\,\dfrac{\Gamma \vdash A}{\Gamma \vdash A, B}}{\Gamma \vdash A \vee B}$$

This can lead to the definition of highly complex inference rules, more difficult to deal with, but which have benefits when it comes to the definition of *normal forms* to represent classes of proofs considered as equivalent by one proof which is designated as the canonical representant [Cha08]. We will always use the notation with a double inference line, as above, to indicate that a rule is a synthetic inference rule — or any compound inference step.

Although derivations are the central object of the theory, most of the work done in the field has been concentrated on a particular class of derivations, those having no premises, thus representing the result of a completed proof construction process.

Definition 1.8. *A* proof *is a derivation where all leaves are axiomatic instances.*

The final layer to define here is the representation of the logical setting, defining which deductive steps are considered valid and which ones are not. Just as Hilbert systems are defined by a set of axioms, a system in structural proof theory is defined by the set of inference rules. The connection to the more general notion of *logic* — as a set of theorems, being a subset of all propositions that can be built on a given language — goes through the proof construction process, since one has to build a proof for a given proposition using only this set of inference rules to show that it is indeed a valid theorem.

Definition 1.9. *A* proof system $\mathscr{S}$ *is a set of inference rules, and it is a system for a given logic* $\mathscr{L}$ *when for any formula* A *in the language of* $\mathscr{L}$*, there is a proof of* $\vdash A$ *in* $\mathscr{S}$ *if and only if* A *is a theorem of* $\mathscr{L}$*.*

A derivation is described as a *derivation in the system* $\mathscr{S}$ if it is built only using inference rules from the $\mathscr{S}$ proof system. Notice that there are two directions in the correspondence between a proof system and some logic. In the first one, where the existence of a proof for the formula A in the system $\mathscr{S}$ implies that A is a theorem of the logic $\mathscr{L}$, we write that $\mathscr{S}$ is *sound with respect to* $\mathscr{L}$. In the other direction, where there exists a proof of A in $\mathscr{S}$ for any given theorem A of $\mathscr{L}$, we write that $\mathscr{S}$ is *complete with respect to* $\mathscr{L}$. Since any proof system defines a logic, as the set of all formulas for which there one can build a proof, these notions of soundness and completeness can also be used between proof systems.

Natural deduction and sequent style. The definition given here for inference rules, and the use of trees to represented derivations, is common grounds for most of traditional logical formalisms. Restrictions can then be imposed to obtain good properties, and for example the sequent calculus formalism emphasises the use of *analytic* inference rules, those where all formulas in the premises are subformulas of formulas in the conclusion — and usually, only the *cut* rule does not conform to this standard, and is therefore eliminated[2]. The characteristic style of the sequent calculus is to use *left rules* and *right rules*, as for example:

$$\wedge_{\mathsf{L}} \frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \qquad \text{and} \qquad \wedge_{\mathsf{R}} \frac{\Gamma \vdash A, \Sigma \quad \Delta \vdash B, \Omega}{\Gamma, \Delta \vdash A \wedge B, \Sigma, \Omega}$$

[2] *Do not attempt this at home!*, it is only formal logic, really.

In natural deduction, deduction is instead focused on the succedent of sequents, and the characteristic style of this formalism[3] is to use so-called *introduction rules* and *elimination rules*, as for example in intuitionistic logic:

$$\wedge_{\mathsf{i}} \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \qquad \text{and} \qquad \wedge_{\mathsf{e}} \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$$

Beyond these differences, the same generic observations and techniques can be used to study the structure of derivations in both the sequent calculus and natural deduction.

1.3 Logical Flow and the Structure of Proofs

The step from proof theory, in the tradition of Hilbert and Gödel, to *structural* proof theory in the style of Gentzen and Prawitz is to consider in details the structure of derivations, and in particular what rules are used, on which *occurrences* of formulas, to find out possible transformations of this structure. This has become a center of interests in many works dedicated to fine-grained proof systems.

A useful tool in this regard is the notion of *logical flow graph* [Bus91], an idea that can also be found in studies of *substructural logics* such as linear logic [Gir87], used to track down particular occurrences of a formula and study how it spreads in a derivation through the application of inference rules. Following the definition given by Buss, we consider occurrences[4] of formulas or subformulas.

Definition 1.10. *A* particle $\underline{A}$ *of a derivation $\mathscr{D}$ is an occurrence of a formula or of a subformula A that appears in a sequent within $\mathscr{D}$.*

In order to use these occurrences, we will have to compare them, but there is a difficulty if two occurrences of the same formula appear in the same sequent, since there is no fixed order in a multiset. Therefore, we consider that a given rule instance attributes an *index* $i \in \mathbb{N}$ to each formula and subformula in its premises and conclusion, which respects the scheme given by the inference rule. This index is a part of the information that one can extract from a given particle.

Example 1.11. *Consider the following instances in the classical sequent calculus:*

$$\mathsf{cont} \frac{(a_1 \vee b_2)_4, (a_1 \vee b_2)_4 \vdash c_3}{(a_1 \vee b_2)_4 \vdash c_3} \qquad \vee_{\mathsf{L}} \frac{a_1 \vdash c_3 \quad b_2 \vdash c_3}{(a_1 \vee b_2)_4 \vdash c_3}$$

Indexes in the contraction instance are used twice in the premise to reflect the scheme of the cont *rule, which specifies that a formula is duplicated, while in the $\vee_{\mathsf{L}}$ instance one formula disappears and its subformulas are promoted into formulas. Notice that indexes do not refer to separate occurrences but to the scheme of inference rules.*

Definition 1.12. *The* basis *of a particle $\underline{A}$ is denoted by $\|\underline{A}\|$ and defined as the pair of the formula A and the index of $\underline{A}$ in the instance where it appears.*

[3] When described with the sequent notation, natural deduction can also be considered as particular form of sequent calculus, alhough the dynamic of proofs is usually described without cut.

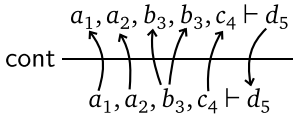
[4] What we call particles corresponds to what Buss calls *s-formulas* in his paper [Bus91].

Now, we want to describe the flow of formulas within a rule instance, and as in the original flow graph definition there are two possible directions for the flow of a formula, depending on the position of the formula in the antecedent or succedent.

Definition 1.13. *Given a rule instance* r*, a particle* $\underline{A}$ *is said to be the* father *of another particle* $\underline{B}$*, which is denoted by* $\underline{A} \succ \underline{B}$*, if and only if* $\|\underline{A}\| = \|\underline{B}\|$ *and either:*

- $\underline{B}$ *appears in the succedent of the conclusion of* r *and* $\underline{A}$ *in one of its premises,*
- *or* $\underline{A}$ *appears in the antecedent of the conclusion of* r *and* $\underline{B}$ *in one of its premise.*

Example 1.14. *Here is a rule instance in the sequent calculus for classical logic, with the father relation illustrated by arrows oriented from a father to its child:*



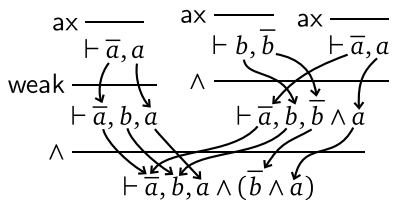
The aggregation of all the arrows representing the father relation within all the rule instances used in a derivation forms the graph we are interested in to represent the influence of applying rules on occurrences of formula.

Definition 1.15. *The* flow-graph *of a derivation* $\mathscr{D}$*, denoted by* $\mathscr{F}(\mathscr{D})$*, is the directed graph* $\langle \mathscr{V}, \mathscr{E} \rangle$ *such that:*

- $\mathscr{V}$ *(nodes) is the set of all particles in* $\mathscr{D}$*,*
- $\mathscr{E}$ *(edges) is such that there is an edge from* $\underline{A}$ *to* $\underline{B}$ *if and only if* $\underline{A} \succ \underline{B}$ *in* $\mathscr{D}$*.*

The graph of a derivation $\mathscr{D}$ is similar to a forest, in the graph-theoretical sense, but the *flow* of some particle — the flow of $\underline{A}$ is the connected subgraph of $\mathscr{F}(\mathscr{D})$ in which $\underline{A}$ appears — is not always a tree. Note that these graphs are not exactly the same as the logical flow-graphs defined by Buss, as they do not track occurrences associated by rules such as the cut and identity rules, for example. Given a proof system, we could extend our graphs into *continuous flow-graphs* that would connect the flows of such associated formulas.

Example 1.16. *Here is the flow-graph for a small proof in the sequent calculus for classical logic, where contraction is built inside the conjunction rule and weakening is used independently:*



Notice that not all edges of the graph are represented, since there is also a connection between the particles corresponding to right part of the decomposed conjunction in the lower $\wedge$ *instance — that is, only connections between atomic particles are shown.*

Using the flow-graph of a derivation, we can observe the relationship between occurrences of formulas, but also between the inference rule instances where these occurrences appear. However, not all occurrences are relevant when considering a particular rule instance, as most formulas in an arbitrary sequent are left untouched by the application of a rule. We can use the father relation in a given instance to detect which particles are relevant to the properties of this instance.

Definition 1.17. *In a rule instance* r*, a particle* $\underline{A}$ *is said to be* passive *if and only if it is the father or child of exactly one particle* $\underline{B}$*, and* $\underline{A}$ *and* $\underline{B}$ *have the same status — either formula or subformula —, and* $\underline{A}$ *is said to be* active *in any other case.*

The rationale for this definition is that occurrences of formulas not modified by the application of a rule — not broken or moved out of formulas, nor duplicated or erased — are said passive and are ignored when considering a given instance. The others are the particles of interests, and we can use them to study the relationship between the rule instances of a derivation.

Example 1.18. *In the following instance of the* $\wedge_{\mathsf{R}}$ *rule, the antecedent is completely duplicated, so that all particles* g_1*,* a_2 *and* $(b_3 \vee c_4)_5$ *which are occurrences of formulas, not subformulas, are active:*

$$\wedge_{\mathsf{R}} \frac{g_1 \vdash a_2 \quad g_1 \vdash b_3 \vee c_4}{g_1 \vdash a_2 \wedge (b_3 \vee c_4)_5}$$

One of the most important application of this distinction is to count the number of contractions, or duplicating instances, that affect a formula during the inference process, since particles are active in an instance duplicating them. Usually, we only need to count duplications above a given instance in a derivation, as done below.

Definition 1.19. *In a derivation* $\mathscr{D}$*, the* multiplicity *of a rule instance* r *is the number of sinks or sources in the union of the flows of all the active particles of* r *that appear above* r *in* $\mathscr{D}$*.*

This definition can be applied to a single particle $\underline{A}$ as well, by considering the sinks and sources of the flow of $\underline{A}$ are located above the point where it appears.

Example 1.20. *Here is a derivation in the sequent calculus for classical logic, where the multiplicity of the lowest rule instance is 3, because its active particle is affected by one contractive rule instance:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{ax}\,\dfrac{}{\vdash \neg A, A}}{\mathsf{weak}\,\dfrac{\vdash \neg A, A, C}{}} \quad \wedge\,\dfrac{\mathsf{ax}\,\dfrac{}{\vdash A, \neg A} \qquad \vdash B, D}{\vdash A, B, \neg A \wedge D}}{\vdash \neg A, A, A, B, C \wedge (\neg A \wedge D)}}{\mathsf{cont}\,\dfrac{}{\vdash \neg A, A, B, C \wedge (\neg A \wedge D)}}}{\vee\,\dfrac{}{\vdash \neg A, A \vee B, C \wedge (\neg A \wedge D)}}$$

Indeed, in the flow-graph of this derivation, there are three sources connected to the particles $\underline{A}$ *and* $\underline{B}$ *active in the lowest rule instance.*

The flow-graph of a given derivation represents only a part of its structure, but it provides enough information to observe many properties of inference rules and of their instances. In particular, the multiplicity of particles shows how duplication and erasure of formulas are handled in a proof system.

Mutiplicatives and Additives. In formalisms such as natural deduction or the sequent calculus, there are usually many different ways of designing a proof system that is both sound and complete with respect to a given logic. For example, we can have different granularities in the design of inference rules using, or not, synthetic rules, if replacing some rules by a synthetic compound preserves the completeness of the system. This lead to the definition of a wealth of systems in natural deduction and even more in the sequent calculus, many of them having different properties, benefits and disadvantages [NvP01].

There is one particularly important distinction to make between two common treatments of traditional logics, such as intuitionistic and classical logics, but also other ones: the difference between *multiplicative* and *additive* presentations [Gir87], where the multiplicity of rule instances is affected either by specific *structural rules*, in the multiplicative case, or by any logical rule in the additive case.

This can be illustrated in classical logic by the shape given to conjunction and disjunction rules. A system with *one-sided* sequents[5] can have the following rules:

$$\wedge\,\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \qquad\qquad \wedge\,\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}$$

$$\vee\,\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B} \qquad\qquad \vee_1\,\frac{\vdash \Gamma, A}{\vdash \Gamma, A \vee B} \qquad \vee_2\,\frac{\vdash \Gamma, B}{\vdash \Gamma, A \vee B}$$

(multiplicative) *(additive)*

When designing a system for classical logic, we can choose between the possible shapes of inference rules, with an impact on the design of other inference rules, but not on soundness or completeness. Indeed, these presentations are connected through the structural rules of weakening and contraction — rules dealing with the structure of a sequent, not with logical formulas themselves. In the multiplicative presentation, conjunction can be complemented by contraction, and disjunction by weakening, so that we derive additive rules as follows:

$$\mathsf{cont}^*\,\frac{\wedge\,\dfrac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, \Gamma, A \wedge B}}{\vdash \Gamma, A \wedge B} \quad\longrightarrow\quad \wedge\,\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}$$

$$\vee\,\frac{\mathsf{weak}\,\dfrac{\vdash \Gamma, A}{\vdash \Gamma, A, B}}{\vdash \Gamma, A \vee B} \quad\longrightarrow\quad \vee_1\,\frac{\vdash \Gamma, A}{\vdash \Gamma, A \vee B}$$

[5] A sequent is said to be one-sided when its antecedent is empty — logics enjoying enough symmetry, such as classical and linear logics, can be implemented by proof systems using only such sequents.

where cont^* denotes several instances of the contraction rule, and the $\vee_2$ rule can be obtained the same way as $\vee_1$, using a weakening on the other formula. We have a symmetric situation with the additive presentation, where the conjunction rule composed with weakenings allows to derive the multiplicative rule, and similarly for the disjunction rule composed with a contraction:

$$\dfrac{\mathsf{weak}^*\dfrac{\vdash \Gamma, A}{\vdash \Gamma, \Delta, A} \quad \mathsf{weak}^*\dfrac{\vdash \Gamma, B}{\vdash \Gamma, \Delta, B}}{\wedge\dfrac{}{\vdash \Gamma, \Delta, A \wedge B}} \quad \longrightarrow \quad \wedge\dfrac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B}$$

$$\dfrac{\mathsf{cont}\dfrac{\vee_1\dfrac{\vee_2\dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A, A \vee B}}{\vdash \Gamma, A \vee B, A \vee B}}{\vdash \Gamma, A \vee B}}{} \quad \longrightarrow \quad \vee\dfrac{\vdash \Gamma, B, A}{\vdash \Gamma, A \vee B}$$

Interestingly enough, one can define a complete system for classical logic using either the multiplicative presentation, and the structural rules of contraction and weakening, or the additive presentation without structural rules, if the axiom rule is also made additive by embedding a weakening rule, but the system formed with the standard axiom and both multiplicative and additive conjunction and disjunction rules is not complete [Hug10]. This is a good illustration of the subtle equilibrium between inference rules required to design a complete proof system.

1.4 Permutations of Rule Instances

One of the most important evolutions of formal logic, triggered by the introduction of structural proof theory, is the growing study of the shape of proofs and of possible transformations of proof objects. The prominent transformations of cut elimination and detour elimination, in the sequent calculus and natural deduction respectively, are at the heart of this field since their first description by Gentzen [Gen34] — and by Prawitz [Pra65] in the case of classical natural deduction. The cut elimination procedure was introduced for the sequent calculus as a tool, in order to study with a fine granularity the detour elimination procedure defined for natural deduction.

Beyond the interest of proving the redundancy of the cut in the sequent calculus — allowing to prove the consistency of the logic —, the cut elimination procedure, as a proof transformation, was found to be a topic of interest in its own right, with the extension of the computational interpretation of proofs [How80] to the sequent calculus [Her94], and natural deduction system with cuts [Pol04]. This is a purely syntactic proof transformation, mainly based on the technique of *permutations* of inference rule instances.

The idea behind permutations is that given a sequent, different inference rules might be applied, or one rule can be applied in several ways. Therefore, when two such instances are used successively on this sequent, the order is irrelevant, and we can decide to change it. This question has raised a lot of attention in proof theory, since the first studies of permutations by Kleene [Kle52] and Curry [Cur52].

Permutation of rule instances is a general idea, which can take different forms in different contexts, and in particular it is significantly simpler in formalisms based on sequents, such as the sequent calculus, than in a formalism where formulas are directly rewritten, such as the calculus of structures. We can express this idea in its general form by using the notion of replacement of a derivation by another, where instances are used in reverse order. To be able to write this, we use the notation:

$$\mathsf{r}_1;\mathsf{r}_2;\cdots;\mathsf{r}_k\,\frac{\Xi_1\vdash\Omega_1\quad\cdots\quad\Xi_n\vdash\Omega_n}{\Gamma\vdash\Delta}$$

to describe any derivation with conclusion $\Gamma\vdash\Delta$, having all the $\Xi_i\vdash\Omega_i$ sequents as premises, and formed by the designated sequence of k instances: the bottommost one is r_1 and the topmost one r_k. Notice that such a sequence of instances defines completely a derivation. In order to define a notion of permutation, we also need to be able to compare rule instances, in a simple way: we write $\mathsf{r}_1\approx\mathsf{r}_2$ and say that r_1 and r_2 are *similar* rule instances, if they are instances of the same inference rule.

Definition 1.21. *Given a derivation $\mathscr{D}$ formed by a sequence $\mathsf{r}_1;\mathsf{r}_2$ of rule instances, as shown on the left, a* permutation *of $\mathscr{D}$ is a derivation $\mathscr{D}'$ as shown on the right:*

$$\mathsf{r}_2\,\frac{\mathsf{r}_1\,\dfrac{\dfrac{\Xi_0\vdash\Omega_0\quad\cdots\quad\Xi_k\vdash\Omega_k}{\Phi\vdash\Sigma}\quad\cdots\quad\Xi_n\vdash\Omega_n}{\Gamma\vdash\Delta}}{}\qquad\longrightarrow\qquad\mathscr{D}'\,\frac{\Xi'_0\vdash\Omega'_0\quad\cdots\quad\Xi'_m\vdash\Omega'_m}{\Gamma\vdash\Delta}$$

where for any $0\leq i\leq m$ we have $\Xi'_i\vdash\Omega'_i=\Xi_j\vdash\Omega_j$ for some $0\leq j\leq n$, and for any two instances $\mathsf{r}_3,\mathsf{r}_4\in\mathscr{D}'$, if $\mathsf{r}_3\approx\mathsf{r}_1$ and $\mathsf{r}_4\approx\mathsf{r}_2$ then r_4 appears below r_3 in $\mathscr{D}'$.

In other words, a permutation of some derivation $\mathscr{D}$ is in general[6] a derivation $\mathscr{D}'$ with the same conclusion and using a subset of the premises of $\mathscr{D}$ — although theses premises might appear several times in $\mathscr{D}'$ — where the rule instances being permuted appear in reverse order in $\mathscr{D}'$.

Remark 1.22. *The permutation of rule instances is trivially extended to permutations of derivations, by considering them as instances of compound inference rules.*

It should be noticed that different cases correspond to the definition given for a permutation. In particular, some permutations are *reversible* while some others are not — given a derivation $\mathscr{D}$ and some permutation $\mathscr{D}'$ of $\mathscr{D}$, we might not be able to obtain $\mathscr{D}$ as a valid permutation of $\mathscr{D}'$. This happens for example if a rule instance in $\mathscr{D}$ was erased in the permutation, so that we would need to recreate it *ex nihilo* if we were to permute $\mathscr{D}'$ back into $\mathscr{D}$. There are also very simple cases of permutation, where no instances are erased nor introduced.

Definition 1.23. *A permutation of a derivation $\mathsf{r}_1;\mathsf{r}_2$ is said to be* trivial *when it is of the shape $\mathsf{r}_4;\mathsf{r}_3$ with $\mathsf{r}_3\approx\mathsf{r}_1$ and $\mathsf{r}_4\approx\mathsf{r}_2$.*

[6] This is not the most general notion of permutation that one can consider: indeed, we could imagine introducing new instances of the inference rules involved in the permutation anywhere in the resulting derivation, but this would require a much more refined notion of similarity of rule instances to enforce the permutation of the particular instances we want to exchange.

Example 1.24. *Below are shown two examples of permutations of rule instances in the sequent calculus for classical logic. In the first case, we have a trivial permutation, where an instance can permute around one branch of another instance:*

$$\dfrac{\dfrac{\vdash \Gamma, A, B, C}{\vdash \Gamma, A \vee B, C}\vee \quad \vdash \Delta, D}{\vdash \Gamma, A \vee B, \Delta, C \wedge D}\wedge \qquad \longleftrightarrow \qquad \dfrac{\vdash \Gamma, A, B, C \quad \vdash \Delta, D}{\vdash \Gamma, A, B, \Delta, C \wedge D}\wedge \quad \dfrac{}{\vdash \Gamma, A \vee B, C \wedge D}\vee$$

and in the second case, we have a more complex permutation involving the duplication of an instance moving above a contraction. Notice that in this case, the transformation from right to left is difficult to use since it requires a particular configuration where both $\vee$ instances and the weakening appear above the contraction.

$$\dfrac{\dfrac{\vdash \Gamma, A, A, B}{\vdash \Gamma, A, B}\mathsf{cont}}{\vdash \Gamma, A \vee B}\vee \qquad \longleftrightarrow \qquad \dfrac{\dfrac{\dfrac{\vdash \Gamma, A, A, B}{\vdash \Gamma, A, B, A, B}\mathsf{weak}}{\vdash \Gamma, A, B, A \vee B}\vee}{\vdash \Gamma, A \vee B, A \vee B}\vee}{\vdash \Gamma, A \vee B}\mathsf{cont}$$

The possibility of permuting two given rule instances depends on the effect they have on the particles they involve. For example, it is obvious that if an instance r_2 crucially relies on the shape of some particule created by another instance r_1, then this r_2 cannot be permuted down under r_1. In general, this possibility depends on the particles that are active in both rule instances: if no particle is active in both instances, then there is no way for one to block the other.

Proposition 1.25. *In any derivation $\mathscr{D}$, if two rule instances r_1 and r_2 appearing one above the other share no active particle, then they can be trivially permuted.*

This follows from the definition of active and passive particles. Indeed, if they have no active particle in common, then any particle active in the instance on top appears exactly once, unchanged, at the bottom of the lower instance. Therefore, the rewriting performed on it by the top instance can also be performed below the lower instance, and is needed only once.

Example 1.26. *When rule instances share an active particle, this particle is blocking their permutation because its active nature implies that it is not available in the same form in the conclusion of the lower instance or in the premise of the upper instance, as illustrated with the derivation:*

$$\dfrac{\dfrac{\vdash \Gamma, A}{\vdash \Gamma, A, B}\mathsf{weak}}{\vdash \Gamma, A \vee B}\vee$$

Definition 1.27. *The* height *of a derivation is the length of its longest branch.*

It should be noticed that permutations might change the height of a derivation, if one instance is replaced with several. However, by definition, trivial permutations behave well once again, as they cannot increase — nor decrease — the height of a proof, where the two instances are simply exchanged.

2 Standard Intuitionistic Systems

As *intuitionism* already had an important place in formal logic at the time Gentzen introduced the idea of proof tree and related formalisms, because of many attempts at improving the proposals of constructive definitions for the whole of mathematics, both natural deduction and the sequent calculus were defined originally [Gen34] in classical logic and intuitionistic logic. The systems defined there in the intuitionistic setting are now widely considered as the standard systems for intuitionistic logic.

We will present here both of these systems using the sequent syntax previously defined. For the sake of simplicity, we will restrict the logical language considered to the essentials of intuitionism: the implication $\to$ connective, and we also use its unit, the truth symbol. As mentioned before, we start with a countable set of atoms and build formulas from atoms, units and connectives.

Definition 2.1. *The* formulas *of intuitionistic logic are generated by the grammar:*

$$A, B \ ::= \ a \mid \top \mid A \to B$$

Both systems, in natural deduction and the sequent calculus, are based on this syntax for formulas and use the same kind of intuitionistic sequent, where only one formula appear on the right-hand side.

Definition 2.2. *An* intuitionistic sequent *is a sequent of the shape* $\Gamma \vdash A$.

Indeed, a striking point in Gentzen systems is the simplicity in the definition of the proof systems for intuitionistic logic. In the sequent calculus, one can obtain the intuitionistic system LJ from the system LK, by simply restricting its inference rules to the case where all sequents are intuitionistic. This comes from the fact that intuitionistic logic is characterised by the invalidity of structural rules on the left of an even number of implications — or equivalently, on the right of a $\vdash$ symbol. This means that if there is only one formula in the right-hand side of the conclusion in a proof, there is no sequent in this proof with no or several formulas on the right.

Then, there are two ways of using sequents. In natural deduction systems, the inference rules normally follow a vertical symmetry, as they are defined by pairs of *introduction* and *elimination* for each connectives, that is one with the connective in the conclusion and not in the premise, and one with the connective in the premise and not in the conclusion. In the sequent calculus, rules are defined by pairs of *left* and *right* rules for each connective, that is one with the connective in the left-hand side of the conclusion, being decomposed in the premises, and the other one with the connective in the right-hand side, decomposed in the premises.

There are several possible semantics for intuitionistic logic, including the early Brouwer-Heyting-Kolmogorov *realizability interpretation* [Tro03], which associates terms built from pairs and functions to intuitionistic formulas, through an encoding based on the work of Kleene on realizability [Kle45]. One can also use an approach in the style of model theory, with a structure of *frames* equipped by a *forcing relation* as defined by *Kripke semantics* [Kri63]. However, we will only focus on the syntactic description of the logic, and its proof theory in the style of Gentzen, which is mainly centered around the normal forms of proofs, allowing to show in particular that the sequent calculus can be made analytic.

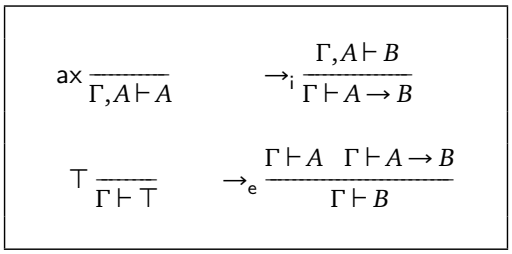


Figure 1: Inference rules for system NJ

2.1 The Natural Deduction System NJ

The natural deduction system for intuitionistic logic is called NJ, for which the set of inference rules is given in Figure 1. Reasoning in such a proof system is based on the intended meaning of formulas, and in the presentation shown here we use the sequent syntax to separate the assumptions from the formula we are proving. The $\top$ rule says that truth can always be proved, and the ax rule says that a formula A is provable if it is given as an assumption. Then, the introduction rule $\to_{\mathsf{i}}$ expresses the relation between implications and assumptions: a formula $A \to B$ is provable if we can prove B with A given as an assumption. Finally, a formula B is provable if we can prove an implication $A \to B$, under the condition that we can also prove this formula A, as embodied by the $\to_{\mathsf{e}}$ rule.

Example 2.3. *Below are shown two proofs in the* NJ *system, where one can notice how the goal formula is extended through elimination instances to match the available hypotheses. The first example is straightforward:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B, B \to A \vdash B}\,\mathsf{ax} \quad \dfrac{}{B, B \to A \vdash B \to A}\,\mathsf{ax}}{B, B \to A \vdash A}\,\to_{\mathsf{e}}}{B \vdash (B \to A) \to A}\,\to_{\mathsf{i}}}{\vdash B \to ((B \to A) \to A)}\,\to_{\mathsf{i}}$$

and the second one requires more manipulations on hypotheses and goals, although it provides an unnecessary hypothesis A, never used in an axiom — to keep the proof readable, we omit some of the hypotheses in several rule instances, since they are not used and are kept only because weakening is performed only within axioms:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots, C \vdash C}\,\mathsf{ax} \quad \dfrac{}{\cdots, C \to B \vdash C \to B}\,\mathsf{ax}}{\cdots, C \to B, C \vdash B}\,\to_{\mathsf{e}}}{\cdots, C \to B, C \vdash A \to B}\,\to_{\mathsf{i}} \quad \dfrac{}{(A \to B) \to A, \cdots \vdash (A \to B) \to A}\,\mathsf{ax}}{(A \to B) \to A, C \to B, C \vdash A}\,\to_{\mathsf{e}}}{(A \to B) \to A, C \to B \vdash C \to A}\,\to_{\mathsf{i}}$$

Remark 2.4. *In standard natural deduction, there is no possibility of permutation of rule instances, because the inference process is directed by the goal formula, the one on the right of the sequent — either in the conclusion, or in the premises as in elimination rules, formulas are always built or decomposed in the succedent.*

There is an important observation to be made about the NJ system. Since it was intented to provide a natural system for reasoning, it allows to use one of the main tools for mathematicians: lemmas, to be used and reused in the proof of theorems, which can be connected to a proof through the implication elimination rule. One can see the right branch of a $\rightarrow_{\mathsf{e}}$ instance as the *main proof* of a formula B, using an hypothesis A, and its left branch as the proof of the lemma A. The mechanism of defining lemmas, which are then used as assumptions, is the basis of the dynamics in natural deduction proofs, since it can be avoided using a proof transformation.

2.2 Detour Elimination

The main result concerning the natural deduction system NJ, on the level of syntax, is the definition of a certain normal form for proof trees, and the proof that the set of these normal forms is complete, in the sense that any given theorem provable in NJ admits a proof of this restricted shape. This notion of a normal form relies on the observation that one can use more implications than necessary in a proof, through the related introduction and elimination rules. We will now concentrate on complete proofs only, rather than on open derivations, that we denote $\mathscr{P}$ — and we write $\mathscr{P} \in \mathsf{NJ}$ if this proof is a valid NJ proof.

Definition 2.5. *A* detour *(on an implication) in a proof $\mathscr{P} \in \mathsf{NJ}$ is the succession of an introduction $\rightarrow_{\mathsf{i}}$ instance and an elimination $\rightarrow_{\mathsf{e}}$ instance, which are sharing their active implication occurrence, as shown below:*

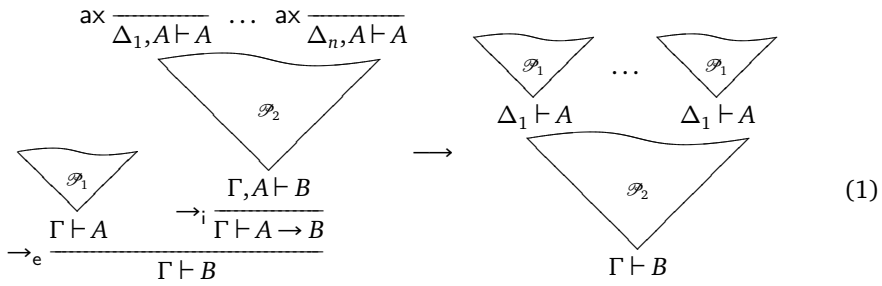
$$\rightarrow_{\mathsf{e}} \dfrac{\Gamma \vdash A \qquad \rightarrow_{\mathsf{i}} \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}}{\Gamma \vdash B}$$

The name *detour* used to describe this construction reflects the idea that it can seem useless to introduce an implication if we are eliminating it immediately. But it has an impact on the shape of the proof, and also on its size, because it creates an assumption A for which only one proof is given, even if this assumption is used to prove several times A. The result of *detour elimination* is expressed in the following theorem, stating that proofs in normal form are sufficient to represent the logic.

Theorem 2.6. *For any proof $\mathscr{P} \in \mathsf{NJ}$ of a formula F, there exists a proof $\mathscr{P}' \in \mathsf{NJ}$ of F in normal form — where no detour on an implication appears.*

There are different ways of proving this result, and in particular we present now two closely related methods, based on syntactic transformations of proofs. The first one is based on a non-local transformation, where the proof of a lemma — a proof of the left premise of an implication elimination in any detour, in our syntax — is directly *plugged* in place of all the axioms corresponding to the use of this lemma, possibly duplicating this subproofs.

Substitutive proof. A *substitutive* procedure can be described by the following scheme, showing how the proof $\mathscr{P}_1$ of the lemma A is plugged in every axiom on A in the main proof $\mathscr{P}_2$, so that the detour can be removed and the assumption A can be erased from the antecedent everywhere in $\mathscr{P}_2$:



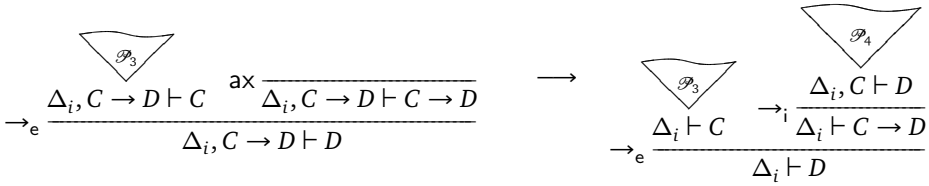
(1)

Note that the axioms on A being replaced are not necessarily all the axioms on any occurrence of A in $\mathscr{P}_2$, but only the axioms where the active particle $\underline{A}$ in the antecedent is in the flow of the active $\underline{A}$ in the introduction instance. Also, the proof built by this transformation contains copies of $\mathscr{P}_1$ and $\mathscr{P}_2$ where the antecedents of all sequents have been modified — in $\mathscr{P}_2$, the assumption on A is not needed anymore, and in a copy of $\mathscr{P}_1$ with conclusion $\Delta_i \vdash A$, all required assumptions are there since $\Gamma \subseteq \Delta_i$ as the rules of NJ only add formulas to antecedents, going up.

The idea of the substitutive method is simply to reduce all detours as described until none is left. The only subtlety here is that the reduction of a detour can create new detours, so that we need to find a better measure than the number of detours in a proof to perform an induction. We will thus use *multiset ordering* [Der82], an ordering that allows to replace a detour with several other detours as long as these new detours have a smaller measure than the original — based on the size of the formula A, denoted by $|A|$ and defined as the number of symbols used in A.

Definition 2.7. *The* rank *of a detour as shown in* (1) *is the size* $|A|$ *of the formula* A *involved, and the rank of a derivation* $\mathscr{D}$ *is the multiset of the ranks of all its detours.*

Proof of Theorem 2.6. By induction on the rank of the given proof $\mathscr{P} \in \mathsf{NJ}$, with a base case when $\mathscr{P}$ is already in normal form, so that the result is trivial. In general, we consider any detour in $\mathscr{P}$, following the scheme given above, where $\mathscr{P}_1$ and $\mathscr{P}_2$ are in normal form. Then, we replace the whole subderivation rooted in this detour by the reduced derivation as shown on the right above. All copies of $\mathscr{P}_1$ and $\mathscr{P}_2$ are of course normal, but the plugging of a copy of $\mathscr{P}_1$ in the place of an axiom might have created a new detour, if A is of the shape $C \to D$, as shown below:



where $\mathscr{P}_4$ is the subderivation above the introduction in $\mathscr{P}_1$. But this new detour has a strictly smaller rank than the eliminated one, because $|C| < |A| = |C \to D|$, and thus all such detours introduced by the elimination of the original detour are of smaller rank. By definition of multiset ordering, the resulting proof has a smaller rank than $\mathscr{P}$ and we can use the induction hypothesis to conclude. □

This proof defines a rather simple normalisation procedure, where the topmost detours are eliminated first, spawning smaller detours that are eliminated following the same process. However, it does not provide any explicit procedure to find which axioms should be replaced or modify all the antecedents of sequents: this rewriting of a proof is *non-local*, in the sense that it requires to inspect arbitrarily large parts of the given proof, and that is not very satisfactory[7].

Permutative proof. Another way of proving that normal forms are enough to have a complete proof system is to use permutations of rule instances. Indeed, as mentioned, a single rule instance in NJ cannot be permuted, but the subderivation defining a detour can be permuted with other rule instances. In order to manipulate a detour in such a way, we define the following compound inference rule:

$$\mathsf{cut}\,\dfrac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \qquad = \qquad \to_{\mathsf{e}}\,\dfrac{\Gamma \vdash A \quad \to_{\mathsf{i}}\,\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}}{\Gamma \vdash B} \tag{2}$$

which is called *cut*, as introduced by Gentzen in the sequent calculus [Gen34]. The idea behind this methodology is to simplify the situation of the substitutive proof by moving detours upwards in a given proof. The crucial observation there is that if the main branch of the detour is reduced to a single axiom instance, then replacing this axiom by the proof of the lemma is trivial:

$$\mathsf{cut}\,\dfrac{\mathscr{P}_1 \quad \Gamma \vdash A \qquad \mathsf{ax}\,\dfrac{}{\Gamma, A \vdash A}}{\Gamma \vdash A} \qquad \longrightarrow \qquad \begin{array}{c}\mathscr{P}_1\\ \Gamma \vdash A\end{array}$$

because there is already a normal proof available for the conclusion of the whole proof, namely the proof of the lemma — assuming we have reduced this proof first.

Remark 2.8. *There is another case where the substitution is trivial, as shown below, but this situation cannot always be reached because an introduction instance affecting A in the proof of the lemma cannot be permuted under a cut.*

$$\mathsf{cut}\,\dfrac{\mathsf{ax}\,\dfrac{}{\Gamma, A \vdash A} \quad \begin{array}{c}\mathscr{P}_2\\ \Gamma, A, A \vdash B\end{array}}{\Gamma, A \vdash B} \qquad = \qquad \begin{array}{c}\mathscr{P}_2\\ \Gamma, A \vdash B\end{array}$$

[7] Mathematically, this proof is enough, but from the viewpoint of computer science, it certainly lacks a clear algorithmic description, that could for example lead directly to an implementation.

Instead of immediately rewriting a detour by a non-local substitution, we can transform it into a cut instance, where the succedent of the conclusion and of the main premise are passive. By proposition 1.25, a cut trivially permutes above most instances but not above an axiom on the lemma A — but might get duplicated.

The permutative proof uses exactly the same technique as the substitutive one, but refines the description by expressing the replacement of the axioms by the proof of the corresponding lemma through the permutation of a cut instance upwards in the proof. As such, it is nothing more than the use of a lemma to prove explicitly that this replacement can be done, but we integrate it into the complete proof for the sake of clarity[8].

Proof of Theorem 2.6. By induction on the rank of the given proof $\mathscr{P} \in \mathsf{NJ}$, just as in the substitutive proof. In the base case, $\mathscr{P}$ is already normal and the result is trivial. In general, we consider any detour in $\mathscr{P}$ following the scheme (1) given for the substitutive proof, where $\mathscr{P}_1$ and $\mathscr{P}_2$ are in normal form. Then, we replace this detour by the corresponding cut instance r and proceed by another induction, on the height of the proof $\mathscr{P}'$ rooted in r, to show that we can transform this $\mathscr{P}'$ into a proof without cut. Each step of the induction consists in a case analysis on r_1, the rule instance above the main premise of r:

1. If r_1 is an instance of $\top$, then we can erase the cut as shown below and use the induction hypothesis, since the cut was erased.

$$\dfrac{\mathscr{P}_1\ \Gamma \vdash A \qquad \top\ \dfrac{}{\Gamma, A \vdash \top}}{\Gamma \vdash \top}\ \mathsf{cut} \quad \longrightarrow \quad \top\ \dfrac{}{\Gamma \vdash \top}$$

2. If r_1 is an instance of ax, we can replace the cut by the proof of the lemma, as shown below, and use the induction hypothesis, since the cut was erased.

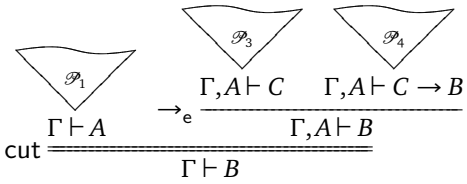
$$\dfrac{\mathscr{P}_1\ \Gamma \vdash A \qquad \mathsf{ax}\ \dfrac{}{\Gamma, A \vdash A}}{\Gamma \vdash A}\ \mathsf{cut} \quad \longrightarrow \quad \mathscr{P}_1\ \Gamma \vdash A$$

3. If r_1 is an instance of the $\to_{\mathsf{i}}$ rule, we can permute the cut above it and use the induction hypothesis, as the height of $\mathscr{P}_3$ is less than the height of $\mathscr{P}_2$.

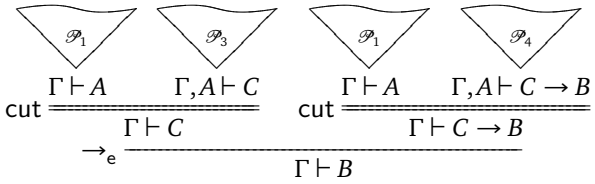
$$\dfrac{\mathscr{P}_1\ \Gamma \vdash A \qquad \to_{\mathsf{i}}\ \dfrac{\mathscr{P}_3\ \Gamma, A, C \vdash D}{\Gamma, A \vdash C \to D}}{\Gamma \vdash C \to D}\ \mathsf{cut} \quad \longrightarrow \quad \dfrac{\dfrac{\mathscr{P}_1\ \Gamma, C \vdash A \qquad \mathscr{P}_3\ \Gamma, C, A \vdash D}{\Gamma, C \vdash D}\ \mathsf{cut}}{\Gamma \vdash C \to D}\ \to_{\mathsf{i}}$$

[8] We are proving that lemmas are not required, after all.

4. If r_1 is an instance of the $\to_{\mathsf{e}}$ rule, we can permute the cut above it but this requires to duplicate the cut, so that from the derivation:



we obtain the new derivation below:



such that we can use the induction hypothesis twice, on both branches of the elimination instance, since $\mathscr{P}_3$ and $\mathscr{P}_4$ have a smaller height than $\mathscr{P}_2$.

In the end of the inductive process, there is no cut instance left in the proof obtained from $\mathscr{P}'$, but it might contain detours that were created in case 2, if the bottommost instance in the proof of the lemma was an introduction instance, and it is plugged on top of an elimination instance — just as in the substitutive proof. However, any such detour has a smaller rank than the detour originally turned into a cut instance, because it is a detour on a subformula of A. By definition of multiset ordering, the complete resulting proof has therefore a rank smaller than the rank of the proof $\mathscr{P}$, and we can use the main induction hypothesis to conclude. □

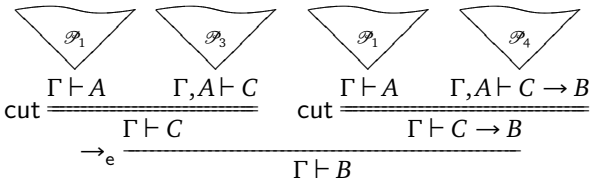
Remark 2.9. *The permutative proof relies on the permutation of cut instances, which are actually derivations considered as compound instances, as permuting a branching instance along one of its branches permutes necessarily the other branch too. But this kind of permutation is allowed, since a whole derivation can permute above another.*

Normalisation order. The goal of this normalisation procedure is to eliminate all detours in a proof, and this is a complex process: we have seen that eliminating a detour can introduce some new detours. From a purely logical point of view, any order in the reduction of detours can be used[9], but from a computational viewpoint this choice can have a huge impact on the efficiency of the process. It is therefore a natural question to ask if at any instant during normalisation, any rewriting step is valid in the sense that it will eventally lead to a normal proof, or if some choice of reduction is » *better* « than another. However, this question is usually approached through computational models rather than in the logical setting, although some of them, such as the λ-calculus, can be directly related to the normalisation process in natural deduction.

[9]The normalisation of natural deduction proofs for intuitionistic logic can be shown confluent, and thus any sequence of rewriting steps leads to the same normal proof.

In the permutative proof given here, the order in which detours are eliminated is important: the one chosen to be next reduced is always one which has no other detour *» above «* in the proofs of its premises. This restriction is necessary to use the simple measure we have used here, because it could not handle a reduction where one of the premise of the cut, containing another detour, would be duplicated. This would make the multiset used as rank grow, and break the induction.

The relaxation of these restrictions is possible if one can find a measure, most likely more complicated than the rank we have defined here, which decreases when a reduction step is performed. A useful tool there is the notion of multiplicity, since it can be used to predict the number of copies of a subproof generated by a detour elimination. For example, in the proof resulting from the rewriting corresponding to a contraction on the cut formula:



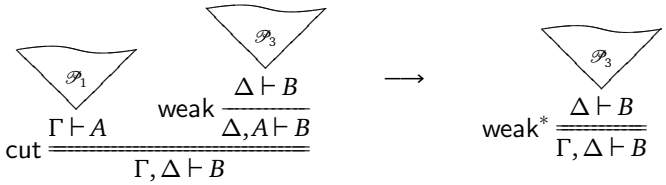
the proof $\mathscr{P}_1$ has been duplicated, along with all the detours it contains. However, if the rank of a detour is not simply the size of the formula involved, but takes into account the multiplicity of the cut that was moved upwards, then it could decrease because the multiplicity of the two copies of the cut is smaller than the multiplicity of the original cut. The problem with this approach is that the multiplicity of all the implication elimination instances *below* some detour must be taken into account, so that the definition of the rank of a derivation is no longer compositional — when a proof with a certain rank is plugged above the left premise of a detour, its rank in the resulting proof is not the same as the original. Moreover, once a proof has replaced an axiom instance, it can be duplicated in the reduction of all the detours involving this axiom. The development of the measure allowing any normalisation order is therefore quite complicated[10], and it would probably require to analyse in a complex way the flow of particles in a given proof.

Structural rules. From a logical perspective, it is unusual to consider variants of the NJ system where structural rules are separated from the other rules, as this means having inference rules modifying the set of assumptions in a sequent. As a consequence, these rules are not goal-directed, creating an ambiguity in the proof construction process: given a sequent, one must decide to apply either a structural rule or another one. This leads to the possibility of permuting instances, so that we could consider several proofs as equivalent — but they would have different sizes, for example. The purpose of such systems is usually to establish a correspondence between proofs and a computational device, such as a variant of the λ-calculus.

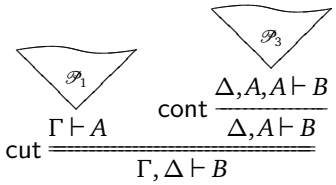
[10]This is not surprising, since the definition of this measure, through the Curry-Howard isomorphism, would provide a direct proof of strong normalisation of the simply typed λ-calculus — which is, rather surprisingly, an open problem, listed for example as Problem #19 in the list of the RTA conference: the usual proof is based on the reducibility candidates technique from Girard.

In the permutative normalisation procedure, the use of structural rules requires the introduction of new rewriting steps, and modifies the rewriting corresponding to a cut moving above an implication elimination instance — in this case, the cut and its lemma is not duplicated, but moved only in one branch of the elimination instance, where the cut formula is required.

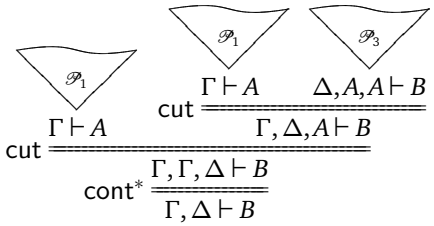
There are two rewriting steps corresponding a cut permuted above a weakening instance. In the most simple case, the formula involved in the weakening is not the cut formula, and the cut is trivially permuted upwards. If the weakening involves the cut formula, then the cut must be erased in the following rewriting:



The permutation of a cut above a contraction instance also corresponds to two possible rewritings. In the first case, as for the weakening, the cut formula is not involved and the cut can simply be permuted upwards. In the principal case, when the cut formula is involved in the contraction, the cut and its lemma are duplicated, so that the following proof:



is turned into the new following proof, which requires new contractions:



The complete proof of detour elimination in the variant of NJ where these rules are used is slightly more complex than the proof given in the basic case, in particular because the rewriting shown above for contraction requires to modifiy the measure of the induction. In this situation, the multiplicity of a cut instance must be a part of its rank, since the bottommost cut instance in the resulting proof involves a formula of the same size as before, and it is located under a proof of the same height as the original. The only difference is that its multiplicity is less than the original.

$$\mathsf{ax}\,\frac{}{A\vdash A} \qquad \mathsf{cut}\,\frac{\Delta\vdash A \quad \Gamma,A\vdash B}{\Gamma,\Delta\vdash B} \qquad \mathsf{weak}\,\frac{\Gamma\vdash B}{\Gamma,A\vdash B}$$

$$\top_{\mathsf{R}}\,\frac{}{\Gamma\vdash\top} \qquad \top_{\mathsf{L}}\,\frac{\Gamma\vdash B}{\Gamma,\top\vdash B} \qquad \mathsf{cont}\,\frac{\Gamma,A,A\vdash B}{\Gamma,A\vdash B}$$

$$\to_{\mathsf{R}}\,\frac{\Gamma,A\vdash B}{\Gamma\vdash A\to B} \qquad \to_{\mathsf{L}}\,\frac{\Delta\vdash A \quad \Gamma,B\vdash C}{\Gamma,\Delta,A\to B\vdash C}$$

Figure 2: Inference rules for system $\mathsf{LJ}\cup\{\mathsf{cut}\}$

2.3 The Sequent Calculus LJ

The sequent calculus for intuitionistic logic is called LJ. The set of inference rules for a particular presentation of its extension $\mathsf{LJ}\cup\{\mathsf{cut}\}$ is given in Figure 2. This is in principle a system similar to the natural deduction system NJ, but where the elimination rule $\to_{\mathsf{e}}$ is replaced by the left rule $\to_{\mathsf{L}}$ for implication — while the right rule $\to_{\mathsf{R}}$ for implication is exactly the same as the $\to_{\mathsf{i}}$ introduction rule. There is also a left rule for the truth unit $\top$, although it is not required if we consider that weakening applies to $\top$. Finally, the structural rules of weakening and contraction are used separately because we have chosen a multiplicative presentation.

The shape of proofs in LJ is quite different from the shape of proofs in NJ, since the antecedent of the sequents is not only used to store assumptions, but contains formulas that can be manipulated through the structural rules and decomposed by the left rules. This means in particular that at each step of the proof construction process, there is a tension between the use of a right rule, dealing with the goal of a given sequent and the use of a left rule to compose or decompose assumptions.

Example 2.10. *Below are shown proofs in the* LJ *sequent calculus, illustrating the use of rules in this system. The first one is not so different from natural deduction:*

$$\dfrac{\dfrac{\dfrac{\mathsf{ax}\,\dfrac{}{B\vdash B} \quad \mathsf{ax}\,\dfrac{}{A\vdash A}}{B,B\to A\vdash A}\to_{\mathsf{L}}}{B\vdash (B\to A)\to A}\to_{\mathsf{R}}}{\vdash B\to((B\to A)\to A)}\to_{\mathsf{R}}$$

but the second has a characteristic shape:

$$\dfrac{\dfrac{\mathsf{ax}\,\dfrac{}{A\vdash A} \quad \dfrac{\mathsf{ax}\,\dfrac{}{B\vdash B} \quad \dfrac{\mathsf{ax}\,\dfrac{}{A\vdash A} \quad \mathsf{ax}\,\dfrac{}{C\vdash C}}{A,A\to C\vdash C}\to_{\mathsf{L}}}{A,B\to(A\to C),B\vdash C}\to_{\mathsf{L}}}{A,A,B\to(A\to C),A\to B\vdash C}\to_{\mathsf{L}}}{A,B\to(A\to C),A\to B\vdash C}\,\mathsf{cont}$$

The sequent calculus LJ is usually presented with the cut rule, as it allows for shorter proofs and can represent detours as found in the proofs in NJ. This rule is exactly the same as the synthetic cut used in natural deduction, but has a status of basic rule in LJ since it cannot be built from more primitive rules in a direct way.

Note that the rule cut is in a way opposite to the axiom rule ax, since the latter says that *» if we have the assumption A, then we can prove A «* while the former says *» if we can prove A, then we have the assumption A «* — that is, together these rules assert the *identity* between A considered as an assumption and A considered as a goal. As we will see, only one of these directions is actually needed, and cut instances can thus be eliminated from a proof.

Remark 2.11. *Since the cut rule of* $\mathsf{LJ} \cup \{\mathsf{cut}\}$ *is the same as the cut used in natural deduction, we use the same vocabulary to describe it, and in particular the formula A in a cut instance* r *as shown below is called the* lemma *involved in* r *and the premise where A is introduced as an assumption is called the* main premise *— and is the root of the* main branch $\mathscr{P}_2$ *of the cut.*

$$\mathsf{cut}\ \dfrac{\overset{\mathscr{P}_1}{\Delta \vdash A} \qquad \overset{\mathscr{P}_2}{\Gamma, A \vdash B}}{\Gamma, \Delta \vdash B}$$

Permutations and normal forms. In natural deduction, the process of detour elimination applied on a proof provides a unique normal form for this proof, since permutations of rule instances are impossible. In the sequent calculus LJ, where both left and right rules are used, these permutations are possible and therefore there we would need to impose some further restrictions in order to obtain a unique normal form, even in this *cut-free* setting.

There are different situations, depending on the considered inference rule. The structural rules are easily permuted, and can be moved to *» canonical «* positions in a given proof. With the weakening rule, there are two possibilities: either instances can be moved upwards to be all located below axiom instances in a proof, or they can be moved down — not all the way, but they can be located immediately above the instance creating or extracting[11] the formula. Notice that the latter defines a more precise normal form, since there is exactly one position for each weakening instance, if the considered proof has only one formula in its conclusion. In the case of the contraction rule, the permutation upwards is not always possible, since this requires the two copies to be *» used the same way «*. The permutation downwards of contraction instances is possible, and this leads these instances to the same location as weakening instances.

The situation of logical left rules is more complicated. Some rules can be easily permuted, such as the $\top_{\mathsf{L}}$ rule, which can be moved either upwards or downwards just as a weakening. The $\to_{\mathsf{L}}$ rule cannot always be permuted, since it performs a splitting of the antecedent of the sequent. These two parts of the antecedent

[11] That is, the rule instance decomposing the formula where a formula appears as a subformula.

must therefore be » *prepared* « and a given $\rightarrow_{\mathsf{L}}$ instance cannot permute below the topmost instance relevant to this preparation. The permutation upwards of an $\rightarrow_{\mathsf{L}}$ instance is easier, since all the modifications on one part of the antecedent can be performed below the $\rightarrow_{\mathsf{L}}$ instance. When permuting such an instance upwards, the limits are the instances affecting the two formulas obtained by the decomposition of the implication. Notice that because this is a branching rule, there are two ways of permuting it upwards: the canonical location for this rule is thus directly below both instances affecting the formulas involved in the $\rightarrow_{\mathsf{L}}$ instance.

Absorbing structural rules. If we can use permutations to move structural rule instances to particular positions in a given proof, then we can simplify the system by allowing these rules to apply *only* at these locations, and then integrate them in synthetic rules. This leads to the definition of variant of $\mathsf{LJ} \cup \{\mathsf{cut}\}$ where the structural rules are built inside other rules, and that we call $\mathsf{LJa} \cup \{\mathsf{cut}\}$ because it follows the principles of additive presentations. The set of inference rules for this system is obtained from $\mathsf{LJ} \cup \{\mathsf{cut}\}$ by removing the weakening and contraction rules, and replacing ax, cut and $\rightarrow_{\mathsf{L}}$ by the following variants:

$$\mathsf{ax}\ \frac{}{\Gamma, A \vdash A} \qquad \mathsf{cut}\ \frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \qquad \rightarrow_{\mathsf{L}}\ \frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma \ni A \rightarrow B \vdash C}$$

where the notation $\Gamma \ni A \rightarrow B$ indicates that the formula $A \rightarrow B$ appears in Γ. Note that in the $\rightarrow_{\mathsf{L}}$ rule, the formula $A \rightarrow B$ is present in both premises — to follow the scheme of » *decomposition* «, it could be removed from the right premise, but it can actually not be removed from the left premise without losing completeness of the system, except if we replace it by a particular set of rules [Dyc92]. This system is not completely additive, because the formula C cannot be duplicated in the $\rightarrow_{\mathsf{L}}$ rule, but this is as close as one can get in intuitionistic logic.

There are other ways of designing variants of the $\mathsf{LJ} \cup \{\mathsf{cut}\}$ system by playing on structural rules and the mutiplicative and additive presentation of other rules. In particular, one interesting possibility is to absorb both weakening and contraction in into the branching rules, as can be done in classical logic to obtain a » *minimal* « calculus [Hug10]. We will be interested in the variant of this *blended* presentation where the weakening rule is kept and the contraction removed, while the axiom rule is kept as in LJ and the cut and $\rightarrow_{\mathsf{L}}$ rules are replaced by the following variants:

$$\mathsf{cut}\ \frac{\Gamma, \Delta \vdash A \quad \Delta, \Psi, A \vdash B}{\Gamma, \Delta, \Psi \vdash B} \qquad \rightarrow_{\mathsf{L}}\ \frac{\Gamma, \Delta \vdash A \quad \Delta, \Psi, B \vdash C}{(\Gamma, \Delta, \Psi) \ni A \rightarrow B \vdash C}$$

where the notation $\Gamma \ni A \rightarrow B$ indicates that the formula $A \rightarrow B$ appears in *one* of the three multisets Γ, Δ and Ψ. This particular variant of the intuitionistic sequent calculus is called $\mathsf{LJb} \cup \{\mathsf{cut}\}$ here. This system is of course also complete for intuitionistic logic, but it allows to avoid the duplication of formulas that are not needed in the antecedent of both premises. These small structural differences will of course have an important impact on possible rule permutations, and on the cut elimination procedure.

2.4 Cut Elimination

The heart of the syntactic study of the sequent calculus is the proof transformation used to show that lemmas are not required, just as in natural deduction. Since the use of lemmas is available in LJ only through the cut rule, this transformation is called *cut elimination*, and it consists, just as detour elimination, in the replacement of an axiom using a lemma by the proof of this lemma. Since the characterisation of normal forms is easier in the sequent calculus than in natural deduction — the equivalent of a detour is always expressed in the form of a cut instance —, the cut elimination theorem is also simpler to express.

Theorem 2.12. *For any proof $\mathscr{P}$ of F in $\mathsf{LJ} \cup \{\mathsf{cut}\}$, there is a proof $\mathscr{P}'$ of F in LJ.*

In other words, cut elimination is simply the result stating the completeness of the cut-free system LJ, with respect to the $\mathsf{LJ} \cup \{\mathsf{cut}\}$ system. In order to prove this, we will define a procedure transforming a proof with cuts into a cut-free proof, but this procedure cannot be as simply described as the substituve procedure used in natural deduction. Indeed, the assumption introduced in the antecedent in the main premise of a cut is not necessarily used as is by an axiom instance, but it can be decomposed into smaller formulas which are then used in axiom instances. We will thus follow the scheme of the permutative procedure given for NJ, and show how cut instances can be moved upwards until they disappear.

As mentioned in the case of natural deduction, the order in which the cuts are eliminated is important, and the situation is even more complicated in $\mathsf{LJ} \cup \{\mathsf{cut}\}$ because contractions can appear freely at any location and there are at any moment possibly several cuts in a proof — a global measure taking all the cuts into account would thus possibly be modified in many ways when a single cut is modified. This leads us to use the same scheme as in natural deduction, eliminating a topmost cut separately from the rest of the proof, and so on until no cut remains. The measure used at the level of the given proof is thus rather simple, but the induction required to eliminate one cut instance is based on a more complicated measure.

Definition 2.13. *The* rank *of a proof in $\mathsf{LJ} \cup \{\mathsf{cut}\}$ is the number of cuts it contains.*

Before we can give the proof describing the cut elimination procedure, we need to prove a lemma stating that the rule $\rightarrow_{\mathsf{R}}$ is *invertible*, so that it can be permuted to the bottom of a proof or to the instance introducing the decomposed formula.

Lemma 2.14. *If $\Gamma \vdash A \rightarrow B$ is provable in LJ then $\Gamma, A \vdash B$ is provable in LJ too.*

Proof. By induction on the height of a given proof $\mathscr{P}$ of $\Gamma \vdash A \rightarrow B$ in LJ, with a base case when the bottommost rule instance r in $\mathscr{P}$ is either a $\rightarrow_{\mathsf{R}}$ instance — in which case we can use the proof above r and we are done — or an axiom, in which case we replace it with the following proof:

$$\mathsf{ax}\,\frac{}{A \rightarrow B \vdash A \rightarrow B} \qquad \longrightarrow \qquad \rightarrow_{\mathsf{L}}\,\frac{\mathsf{ax}\,\dfrac{}{A \vdash A} \quad \mathsf{ax}\,\dfrac{}{B \vdash B}}{A \rightarrow B, A \vdash B}$$

In the general case, we can always rewrite the conclusion of the proof above the bottommost instance r and use the induction hypothesis on the proof above. □

Proof of Theorem 2.12. By induction on the rank of the given proof $\mathscr{P} \in \mathsf{LJ} \cup \{\mathsf{cut}\}$, with a base case when $\mathscr{P}$ is already cut-free, and is a valid proof in LJ. In general, we consider a cut instance r in $\mathscr{P}$ connecting some lemma A, such that the proofs above both premises of r are cut-free, and use an induction on the triple $(|A|, m, h)$ under lexicographic order, where m is the multiplicity of r in $\mathscr{P}$ and h is the height of the main branch $\mathscr{P}_2$ of r — to show that the proof rooted in r can be transformed into a cut-free proof. At each step, we use a case analysis on the rule instance r_1 at the root of $\mathscr{P}_2$, above the main premise of r:

$$\dfrac{\mathscr{P}_1 \quad \Delta \vdash A \qquad \mathsf{r}_1\dfrac{\mathscr{P}_2 \quad \Phi \vdash C}{\Gamma, A \vdash B}}{\mathsf{cut}\dfrac{}{\Gamma, \Delta \vdash B}} \qquad \text{or} \qquad \dfrac{\mathscr{P}_1 \quad \Delta \vdash A \qquad \mathsf{r}_1\dfrac{}{\Gamma, A \vdash B}}{\mathsf{cut}\dfrac{}{\Gamma, \Delta \vdash B}}$$

1. If r_1 is a left rule affecting only particles in Γ or a right rule affecting B, then it can permute below the cut either by Proposition 1.25, or by replacing the whole proof with r_1 if it was a $\top_{\mathsf{R}}$ instance — in the first case, we can use the induction hypothesis since the height of the main branch has decreased, and in the second case we are done.

2. If r_1 is an axiom instance on A, then the whole proof can be replaced with the proof $\mathscr{P}_1$ of the lemma A and we are done, since the result is cut-free:

$$\dfrac{\mathscr{P}_1 \quad \Delta \vdash A \qquad \mathsf{r}_1\dfrac{}{A \vdash A}}{\mathsf{cut}\dfrac{}{\Delta \vdash A}} \quad \longrightarrow \quad \begin{array}{c}\mathscr{P}_1 \\ \Delta \vdash A\end{array}$$

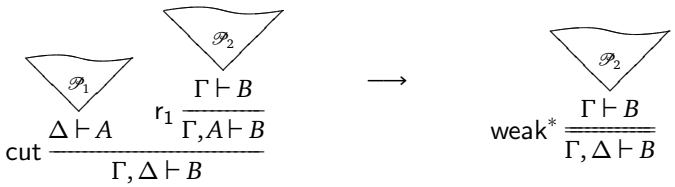
3. If r_1 is a left $\to_{\mathsf{L}}$ instance decomposing the particle $\underline{A}$ in the main premise of r, then we can use the invertibility of $\to_{\mathsf{R}}$ stated in Lemma 2.14 and rewrite the proof of the lemma $A = C \to D$ as shown below on the left, so that the cut instance can be splitted into two new cut instances used sequentialy, first on C and then on D, as follows:

$$\dfrac{\mathscr{P}_5 \quad \Delta, C \vdash D}{\to_{\mathsf{R}}\dfrac{}{\Delta \vdash C \to D}} \quad \dfrac{\mathscr{P}_3 \quad \Sigma \vdash C \qquad \mathscr{P}_4 \quad \Omega, D \vdash B}{\mathsf{r}_1\dfrac{}{\Omega, \Sigma, C \to D \vdash B}} \quad \mathsf{cut}\dfrac{}{\Omega, \Sigma, \Delta \vdash B} \qquad \searrow$$

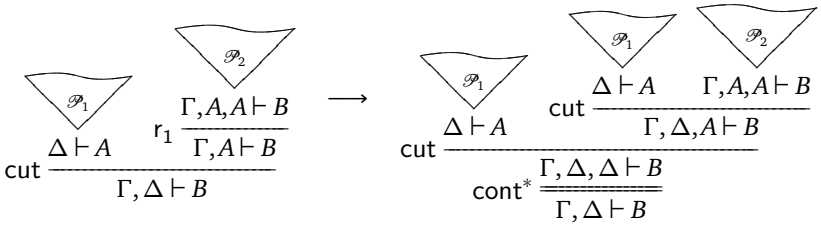
$$\dfrac{\mathscr{P}_3 \quad \Sigma \vdash C \qquad \dfrac{\mathscr{P}_5 \quad \Delta, C \vdash D \qquad \mathscr{P}_4 \quad \Omega, D \vdash B}{\mathsf{cut}\dfrac{}{\Omega, \Delta, C \vdash B}}}{\mathsf{cut}\dfrac{}{\Omega, \Sigma, \Delta \vdash B}}$$

and we use the induction hypothesis, first on the upper cut and then on the lower cut, since their active particles have bases smaller in size than $|A|$ — indeed, we have $A = C \to D$ and therefore $|C| < |A| = |C \to D|$, and the same holds for $|D|$.

4. If r_1 is a weakening instance erasing the particle $\underline{A}$ in the main premise of r, then the cut and the proof of the lemma are erased, and weakening instances are introduced to erase the assumptions in Δ — and we are done with the induction, since the result is cut-free:



5. If r_1 is a contraction instance duplicating the particle $\underline{A}$ in the main premise of r, then the cut and the proof of the lemma A are duplicated, used sequentially, and contractions are introduced to duplicate the necessary assumptions:



   and we can use the induction hypothesis first on the upper cut and then on the lower cut, since their multiplicity is less than the multiplicity h of r, the orginal cut instance, while the size of the active particle $\underline{A}$ in unchanged.

In the end of the inductive process, the proof rooted in r has been replaced with a cut-free proof of the same sequent $\Gamma, \Delta \vdash B$, and the rank of the resulting proof is thus less than the rank of the original proof $\mathscr{P}$. By induction hypothesis, we obtain a cut-free proof $\mathscr{P}'$ of same conclusion as the given proof $\mathscr{P}$. □

Although it takes place in a setting where » *detours* « are all represented by cut instances, the proof of cut admissibility above defines a cut elimination procedure quite similar to the procedure used in NJ. One should however observe that the situation after the elimination of one cut is simpler than after the elimination of a detour in NJ, since cut elimination cannot introduce new cuts.

Elimination order. As in detour elimination in natural deduction, the order in which cuts are eliminated in the proof described above can have a great impact on the computation required to obtain a normal form. Topmost cuts were eliminated first here, which is a quite constrained setting, but the ability to eliminate cuts in a completely different order comes at the same price as in natural deduction: the measure required to control the induction will be much more complicated. Using somehow the notion of multiplicity is required if the contraction rule is used, but it must be used with care if we attempt to eliminate a cut which has other cuts in its premises. For example, in the case of a cut moving above a contraction, and being

duplicated, the multiplicity of this one cut decreasese, but it can increase for other cuts below, since a subproof is duplicated, and new contractions are introduced. In order to define such a valid measure, one should thuse take into account not only the contractions obviously duplicating the cut formula, but also the ones that might appear during the elimination process of all the other cuts — in other words, the terminating nature of cut elimination is a global property of the proof structure.

It should also be noticed that if cuts are eliminated in a different order, possibly step by step in an *interleaved* way, we have to be able to move a cut above another cut, and this should not affect the induction measure, since this is actually more an exchange than a reduction — it can be done one way or the other, depending on which cut should appear at the top. The two following derivations should therefore have the same measure:

$$\dfrac{\dfrac{\mathscr{P}_1}{\Gamma \vdash A} \quad \dfrac{\mathscr{P}_2 \quad \mathscr{P}_3}{\Delta, A \vdash B \quad \Psi, A, B \vdash C}\,\mathsf{cut}}{\Gamma, \Delta, \Psi \vdash C}\,\mathsf{cut} \qquad \dfrac{\dfrac{\mathscr{P}_2}{\Delta \vdash B} \quad \dfrac{\mathscr{P}_1 \quad \mathscr{P}_3}{\Gamma, B \vdash A \quad \Psi, A, B \vdash C}\,\mathsf{cut}}{\Gamma, \Delta, \Psi \vdash C}\,\mathsf{cut}$$

Structural variations. The different systems that can be obtained by modifying the rules of $\mathsf{LJ} \cup \{\mathsf{cut}\}$ admit a similar proof of cut elimination. However, all their structural differences have an impact on the erasure and duplication of cuts, as can be seen in the measure needed to prove cut elimination. The $\mathsf{LJa} \cup \{\mathsf{cut}\}$ system is much simpler than $\mathsf{LJ} \cup \{\mathsf{cut}\}$ in this regard, since the duplication performed in the branching rules never creates two copies of a formula in the same branch. This means that the multiplicity of cuts is not needed in the definition of the rank of derivations, because the height of the proof above a cut decreases after rewriting:

$$\dfrac{\dfrac{\mathscr{P}_1}{\Gamma \vdash A} \quad \dfrac{\dfrac{\mathscr{P}_3 \quad \mathscr{P}_2}{\Gamma, A \vdash C \quad \Gamma, A, D \vdash B}\,\mathsf{r}_1}{\Gamma, A \vdash B}\,\mathsf{cut}}{\Gamma \ni C \to D \vdash B}$$

into the derivation:

$$\dfrac{\dfrac{\mathscr{P}_1}{\Gamma \vdash A} \quad \dfrac{\mathscr{P}_3}{\Gamma, A \vdash C}\,\mathsf{cut}}{\dfrac{\Gamma \vdash C}{\Gamma \ni C \to D \vdash B}\,\mathsf{r}_1} \qquad \dfrac{\dfrac{\mathscr{P}_1}{\Gamma, D \vdash A} \quad \dfrac{\mathscr{P}_2}{\Gamma, A, D \vdash B}\,\mathsf{cut}}{\Gamma, D \vdash B}\,\mathsf{cut}$$

in the case of a cut moving above an implication left instance. Both copies of the cut — along with the proof $\mathscr{P}_1$ of the involved lemma — are then located under a proof, either $\mathscr{P}_2$ or $\mathscr{P}_3$, which is smaller than their original right premise.

3 Deep Inference and Nested Proof Systems

The notion of *deep inference* [Gug07] is a relatively recent[12] methodology providing a uniform approach to structural proof theory, based on the postulate that the rules of deduction, such as the inference rules used in traditional proof systems, can be applied anywhere *inside* a formula. As a consequence of this new way of performing deduction, it is necessary to find a proper treatment of the logical *meta-level* — the *apparatus* of multisets, sequents and branches used in standard systems.

This idea of applying inference rules deep inside formulas was not consistently developped up to the form of a logical formalism radically generalising formalisms using sequents before the introduction of the *calculus of structures* [Gug07], but it can be found in one form or another in many earlier studies. Already in the work of Schütte [Sch60], inference rules are defined that can apply *» inside positive and negative parts of a formula «*, where these parts are considered as a generalisation of the notions of antecedent and succedent in the sequents[13]. The closest precursor of deep inference might be *display logic* [Bel82], designed to handle a wide variety of logics, where subformulas can be accessed through the use of a kind of intermediate level between sequents and formulas. However, this requires to *» zip and unzip «* formulas during the deduction process, and does not support all kind of inference rules and interactions between subformulas. Another example of related system is the one given by Pym for the logic BI [Pym02].

The goal of the deep inference methodology is to accept all the consequences of applying inference rules inside formulas and study the benefits of the new setting obtained this way. The first consequence of this idea is that the result of applying an inference rule on a formula A inside some formula B must itself a formula, that is plugged in the place of A in B. From a sequent-based perspective, this means that the metal-level syntax allowing to write for example $C, D \vdash E$ must be valid at the level of formulas. This yields the founding principle of the deep inference view of proof objects as usually described in proof systems, which can be summarised as follows:

Turnstiles, commas and spaces are just another syntax for connectives, and deduction is the logically consistent rewriting of a formula into another formula.

Deduction as rewriting. This approach offers a highly syntactic way of viewing and manipulating formulas and proofs, since sequents — which were defined above using multiset in intuitionistic natural deduction and sequent calculus — are seen as mere syntactic constucts following a given grammar, just as formulas. Although it can seem unusual from the perspective of traditional formalisms, it corresponds to the intuition that for example, in the classical sequent calculus, commas on the left represent conjunctions, commas on the right represent disjunctions, while the turnstile $\vdash$ represents an implication.

[12] The development of the idea of deep inference, in the form of the calculus of structures and other formalisms due to Guglielmi, can actually be traced back to [Gug99], but the initial article describing this approach suffered from a rather long editorial process.

[13] As mentioned in [Brü03] ; the generalisation of antecedents and succedents is also relevant to the merging of logical and meta-level syntaxes used in the calculus of structures.

When everything is seen as a syntactic construct, deduction can be reduced to the rewriting process turning a construct into another. But not any rewriting rule is acceptable, since it must represent a valid deduction step. Therefore, given for example two syntactic constructs $\Gamma \vdash A$ and $\Gamma \vdash B$, the rule instance:

$$\mathsf{r}\,\frac{\Gamma \vdash A}{\Gamma \vdash B}$$

is valid in a deep inference system only if the construct $\Gamma \vdash A$ implies[14] the construct $\Gamma \vdash B$ in the logic represented. The situation might seem different in the case of a branching rule, but it can also fit the rewriting scheme if *» spaces between branches «* are considered as connectives. This is done by considering the connective implictly used between two branches, for example in the classical sequent calculus, as in the following rule instance:

$$\wedge_{\mathsf{R}}\,\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \text{corresponds to} \qquad \wedge_{\mathsf{R}}\,\frac{(\Gamma \vdash A) \wedge (\Gamma \vdash B)}{\Gamma \vdash A \wedge B}$$

From this perspective, a deduction step simply replaces some logical expression, a formula, with another formula. And as any rewriting process, there is no reason to restrict this transformation to the outer level of an expression, as illustrated by the rule instance shown below, valid in a system if C can be deduced from A:

$$\mathsf{r}\,\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash C \wedge B} \qquad \text{is written} \qquad \mathsf{r}\,\frac{\xi\{A\}}{\xi\{C\}} \qquad \text{where } \xi\{-\} = \Gamma \vdash - \wedge B$$

In the following, we will call *nested proof system* any system that allows to apply an inference rule on any formula nested inside another formula.

Expressivity. The deep inference methodology was originally introduced with the purpose of defining a logical formalism that would support the representation of the logic BV [Gug07], which uses a self-dual, non-commutative connective and is similar to the *pomset logic* [Ret97] defined previously by Retoré. The problem of this logic was the difficulty to obtain a sequent calculus representing it, as already pomset logic uses a proof theory based on some variant of proof-nets [Gir96], but no suitable presentation in the sequent calculus was ever found. It later turned out that BV cannot be represented in the sequent calculus, because the level of depth allowed in the application of inference rules in a system for BV cannot be restricted without restricting the logic [Tiu06b]. In this sense, nested formalisms such as the calculus of structures are strictly more expressive than traditional formalisms based on the shallow application of inference rules, as natural deduction and the sequent calculus.

Remark 3.1. *Although the* BV *logic is an example of the need for deep inference, the precise difference in expressive power between, for example, the calculus of structures and the sequent calculus, is still unclear — and* BV *itself is not yet well understood, as in particular it is not known whether it coincides with pomset logic or not.*

[14] A notion of implication is always present, in any logical system, as the embodiment of the notion of deduction — without it, a logic is a set of unrelated theorems, and should not even be called a logic.

This expressivity can also be seen in the area of *modal logics*, since many such logics, as for example S5, have no well-behaved sequent calculus where the cut rule is admissible. Different extensions of the sequent calculus such as the *hypersequent calculus* [Avr96], various *labelled calculi* [Vig00] and display logics have been used, but the deep inference approach allows to define systematically well-behaved proof systems while staying inside of the language of modal logic [Brü10].

Finally, there is an orthogonal question in the structural representation of logics, which has become a center of interest in the *post-linear-logic* studies of traditional logics: the level of decomposition of the deduction process that allows, or not, for certain analyses. In this regard, the deep inference methodology leads naturally to the definition of fine-grained inference rules — the stereotypical example is the *switch* found in virtually any system in the calculus of structures, which decomposes the splitting of assumption contexts in traditional branching rules. Moreover, this decomposition power allows to define *local* proof systems, where an inference rule can erase or duplicate[15] only atomic formulas, by the *decoupling* of contraction and the shuffling abilities required in a deep inference system, through the *medial* rule and its variants [Str07]. Such systems have be designed for classical logic [Brü06c] in the calculus of structures as well as linear logic [Str03a].

3.1 The Calculus of Structures

The most prominent logical formalism using deep inference is probably the *calculus of structures*, which is also the one originally introduced as the embodiment of the idea of applying inference rule deep inside and anywhere in formulas, in the work of Guglielmi [Gug07]. It is a direct, and pure application of the principles described above, and allows for a uniform treatment of many different logics. Throughout the description of this formalism, we will illustrate the notions with the most standard representation of classical logic in this setting.

Syntactic structure. The basic layer of this formalism is exactly the same as in more traditional systems: formulas are generated by a given grammar, and can be seen as syntactic trees where inner nodes are connectives and leaves are units and atoms — picked from a countable set $\mathscr{A}$. For example, classical formulas are generated by conjunction, disjunction and their units.

Definition 3.2. *The* formulas *of classical logic are generated by the grammar:*

$$A, B ::= a \mid \top \mid \bot \mid A \wedge B \mid A \vee B$$

We will use the same naming conventions as before, and we also stay at the level of propositional logic, where no quantifiers are used. Formulas are read as usual, and the behaviour of units with respect to their corresponding connective is not defined by this syntactic definition — in the sequent calculus, that would be defined through inference rules and the meta-level interpretation given for commas. But a formula is not the basic object manipulated at the deduction level in the calculus of structures: as suggested by its name, this formalism manipulates structures.

[15] Atomic weakening is easy to obtain in the sequent calculus, but atomic contraction does not seem to be possible in a shallow setting, since it relies entirely on an essentially » *deep* « shuffling of formulas.

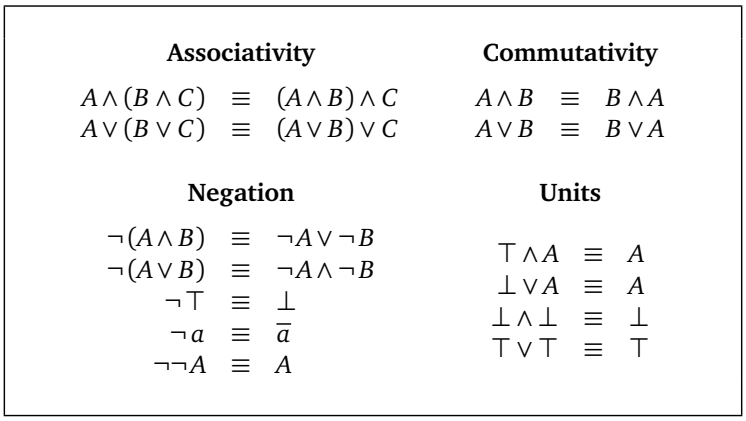


Figure 3: Congruence used on classical formulas

A structure is basically a formula where a certain number of logical equivalences have been built-in to facilitate the inference process and keep notations and proofs as readable as possible — from the perspective of a human reader. When defining a system, the precise equivalences chosen to be part of the *congruence* on which the structures are built depend on the purpose of this system. In particular, a proof system with more equivalences is easier to use for humans, while a system using as few equations as possible is easier to implement on a machine. The equations shown in Figure 3 are used in the most standard system for classical logic.

Definition 3.3. *The* structures *of classical logic are the equivalence classes of classical formulas generated by the contextual closure of the equivalence relation defined by the equations given in Figure 3.*

A formula in the calculus of structures is thus always considered through the glasses of the chosen congruence, and the use of the contextual closure of the given equivalence relation implies that this *» automatic rewriting «* of a formula is used deep inside it and not only at the outer level. For the sake of simplicity, structures are denoted by capital letters such A, B, C, as formulas.

Remark 3.4. *Because of the congruence, a given structure is representing potentially infinitely many different formulas, so that one must decide how to write this structure among infinitely many representations. This is not a problem, because we can consider normal forms, where negation is pushed to the atoms, units are removed as much as possible, a default associativity is chosen for connectives, and a total order is defined over atoms and extend to formulas, to deal with commutativity.*

The level of deduction in the calculus of structures follows the definitions given previously for the general case but it is based on structures rather than formulas, and makes the notion of sequent disappear, by imposing that a sequent is always of the shape $\vdash A$, with an empty antecedent and a succedent consisting of only one formula. Therefore, we will never mention sequents in the calculus of structures, and always consider that inference rules are applied on structures.

The replacement of sequents by structures is a restriction from the viewpoint of the broad definitions given previously, and in natural deduction or in the sequent calculus, it would correspond to the use of inference rules replacing one formula by another. But in this restricted setting, the ability to apply inference rules deep inside formulas allows to encode in these formulas many combinations of sequents, making of the calculus of structures a generalisation of traditional formalisms. The ability to use deep inference relies on the use of schemes, to define inference rules, where a structure can be asserted to be a substructure of any possible structure.

Definition 3.5. *A* context *is a structure with a hole, to be filled by another structure.*

We denote contexts by letters such as ξ, ζ or θ, and a hole is denoted with $-$, so that some particular context can be written for example $\xi\{-\}$, and when this hole is filled with a structure A we can denote by $\xi\{A\}$ the resulting structure. This allows to write inference rules of the shape:

$$\frac{\xi\{A\}}{\xi\{B\}} \qquad \text{possibly written with an implicit context} \qquad \frac{A}{B}$$

where the implicit notation of the context is used for inference rules but never for their instances, since instances correspond to the use of a particular context ξ.

Another restriction applies to inference rules in the calculus of structures: they can only have exactly one premise. This corresponds to the idea that if an inference rule is applied on a structure nested inside another one, then the result needs to be substituted in place of the original substructure. As a consequence, derivations in the calculus of structures are always sequences of rule instances, denoted as shown below on the left, and proofs are derivations from a particular structure $\bullet$ to any structure, as shown on the right — we use $\bullet$ to denote this » *truth* « structure here but it depends on the logic represented.

$$\begin{array}{c} A \\ \mathscr{D}\| \\ B \end{array} \qquad\qquad \begin{array}{c} \bullet \\ \mathscr{P}\| \\ B \end{array} \qquad \text{also denoted} \qquad \begin{array}{c} \mathscr{P}\Vdots \\ B \end{array}$$

The classical KS **system**. As an example, we present the most standard system for classical logic in the calculus of structures, the KS system[16] [Brü03], for which inference rules are given in Figure 4. It is based on classical formulas as we defined above, and structures are defined by the equations given in Figure 3. In this system, the truth unit $\top$ is the structure $\bullet$ used as the premise for proofs, and inference rules are shown with an implicit context, so that they can be applied deeply.

The use of a congruence to deal with negation and units makes this proof system small, with only four inference rules, and easy to read. The rules $\mathsf{i}\downarrow$, $\mathsf{w}\downarrow$ and $\mathsf{c}\downarrow$ can be viewed through the intuitive logical interpretation of sequents as standard sequent rules, in a one-sided version of LK. In this interpretation, $\top$ can represent the empty premise and $\bot$ the empty subset in a multiset, while $\vee$ is a comma.

[16] The system presented here is named KSg in the work of Brünnler, while KS is the local variant, but we will drop the annotations for locality here for simplicity, since we will not present the local system.

$$\mathsf{i}\downarrow\frac{}{A\vee\neg A}\top \qquad \mathsf{s}\frac{(A\vee B)\wedge C}{A\vee(B\wedge C)} \qquad \mathsf{w}\downarrow\frac{}{A}\bot \qquad \mathsf{c}\downarrow\frac{A\vee A}{A}$$

Figure 4: Inference rules for system KS

Although these rules are similar to the standard ones, they can be applied inside structures and therefore derivations such as the following:

$$\mathsf{w}\downarrow\dfrac{\dfrac{\mathsf{i}\downarrow\dfrac{}{B\wedge(A\vee\neg A)}\top}{B\wedge((A\wedge(\top\vee\bot))\vee\neg A)}\equiv}{B\wedge((A\wedge(\top\vee C))\vee\neg A)}$$

are valid in the KS system, where $\equiv$ is a *» fake «* inference rule that we use to make the rewriting performed through the congruence explicit in the representation of a derivation. This can be compared to the way this derivation would be written in a one-sided, multiplicative version of LK:

$$\wedge\dfrac{\vdash B \qquad \vee\dfrac{\wedge\dfrac{\mathsf{ax}\dfrac{}{\vdash A,\neg A} \qquad \vee\dfrac{\mathsf{weak}\dfrac{\vdash\top}{\vdash\top,C}}{\vdash\top\vee C}}{\vdash A\wedge(\top\vee C),\neg A}}{\vdash(A\wedge(\top\vee C))\vee\neg A}}{\vdash B\wedge((A\wedge(\top\vee C))\vee\neg A)}$$

The fourth inference rule s, called the *switch* rule, is the most particular rule in KS and is a specificity of the calculus of structures. It is the only rule in this system to deal with logical connectives, and it does so by defining the *interaction* between conjunction and disjunction. More precisely, it expresses the *weak distributivity* of $\wedge$ over $\vee$, and this is used to implement in small steps the splitting of the context at the heart of the $\wedge$ inference rule, as for example in the derivation:

$$\wedge\frac{\vdash C,A \quad \vdash D,B}{\vdash C,D,A\wedge B} \qquad \text{expressed as} \qquad \mathsf{s}\dfrac{\mathsf{s}\dfrac{(C\vee A)\wedge(D\vee B)}{C\vee(A\wedge(D\vee B))}}{C\vee D\vee(A\wedge B)}$$

where we see that the $\wedge$ connective is not removed in the calculus of structures but kept, representing the space between branches in the sequent calculus. This rule contains all the required treatment of $\vee$, which is simply identified with the comma in the sequent calculus, and is thus involved in the splitting.

Remark 3.6. *The* KS *system can be divided in three fragments with different purposes: the* identity *fragment* $\{\mathsf{i}\downarrow\}$ *deals with the identity of structures, the* structural *fragment* $\{\mathsf{w}\downarrow,\mathsf{c}\downarrow\}$ *deals with the erasure and duplication of structures, and finally the* logical *fragment* $\{\mathsf{s}\}$ *deals with the logical connectives.*

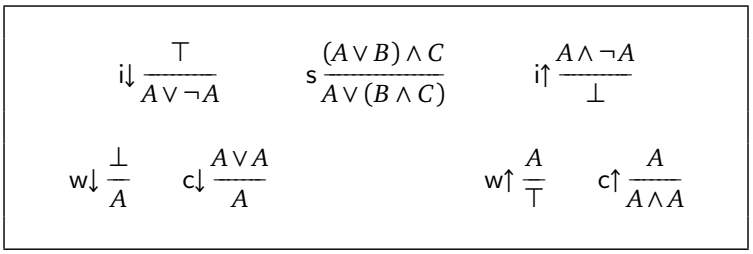


Figure 5: Inference rules for system SKS

The KS system does not enjoy the *subformula property* in its usual sense, but it is analytic in that no new atoms are introduced during proof search. There exists a correspondence, informally present in the description of the rules of KS above, between this calculus and the cut-free sequent calculus LK. The equivalent of the cut rule in this setting would be the *dual* of the identity rule i↓:

$$\mathsf{i}\uparrow \frac{A \wedge \neg A}{\bot} \qquad \text{similar to} \qquad \begin{array}{c} \wedge \dfrac{\vdash \Gamma, A \quad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta, A \wedge \neg A} \\ \text{» cut «}\ \dfrac{}{\vdash \Gamma, \Delta} \end{array}$$

where the dual of an inference rule is the rule obtained by exchanging and dualising premise and conclusion. This notion of *dualisation* of a structure is an involution, but depends on the logic represented: in classical logic it corresponds to negation, but in other logics it might be another operation — for example in logics where negation is not an involution.

The SKS symmetric system. The identity rule i↓ is not the only one of which we consider the dual. The most general, and uniform view of duality in the calculus of structures is to close a system under duality, so that for any rule r↓ in a system the rule r↑ also belongs to this system, and the other way around — this justifies the annotations on rule names by defining two fragments in the system: a rule r↓ is called a *down rule* and belongs to the down fragment, while a rule r↑ is called an *up rule* and belongs to the up fragment. Following this idea, we extend KS into the symmetric system SKS, with the set of inference rules is shown in Figure 5, in a simple notation using implicit contexts.

In the SKS system, all rules have a dual, but the switch rule s does not follow the naming scheme, using annotations with up and down arrows. This is because it is *self-dual* and therefore belongs to both[17] the down and up fragments:

$$\frac{\neg(A \vee (B \wedge C))}{\neg((A \vee B) \wedge C)} \equiv \frac{\neg A \wedge (\neg B \vee \neg C))}{((\neg A \wedge \neg B) \vee \neg C)} \equiv \mathsf{s}\frac{(\neg C \vee \neg B) \wedge \neg A}{\neg C \vee (\neg B \wedge \neg A)}$$

as can be seen here, using dualities of connectives and the fact that the negation is involutive in classical logic. Note that it relies on the commutativity of ∧ and ∨.

[17] The switch is always necessary and thus cannot belong to only one of the fragments; this self-duality illustrate the perfect duality of ∧ and ∨, and of their behaviour, in classical logic.

Since duality can be applied to inference rules and their instances, this can be generalised to complete proofs and derivations. Given some derivation $\mathscr{D}$ in SKS, its dual $\neg\mathscr{D}$ can be obtained by reversing the order of inference rule instances, and applying negation to all structures involved. The dual of a derivation from A to B is thus a derivation from $\neg B$ to $\neg A$.

Example 3.7. *Below are shown three derivations in the symmetric* SKS *system. The two proofs on the left illustrate how a rule of the down fragment can be simulated by its dual and the cut.*

$$\dfrac{\dfrac{B}{\equiv\dfrac{B}{B\wedge\top}}}{\mathsf{w}\!\downarrow\dfrac{}{A\vee(B\wedge\top)}} \qquad \dfrac{\dfrac{B}{\mathsf{i}\!\downarrow\dfrac{}{B\wedge(A\vee\neg A)}}}{\mathsf{w}\!\uparrow\dfrac{}{B\wedge(A\vee\top)}}\,\mathsf{s}\dfrac{}{A\vee(B\wedge\top)} \qquad \dfrac{\dfrac{A}{\mathsf{i}\!\downarrow\dfrac{}{(A\vee\neg A)\wedge A}}}{\mathsf{s}\dfrac{}{A\vee(\neg A\wedge A)}}\,\mathsf{i}\!\uparrow\dfrac{}{A\vee\bot}$$

The KS and SKS systems are the most basic presentation of classical logic in the calculus of structures, but many variants can be designed, and use specific features of the deep inference setting to improve on these. As mentioned, the rule below called *medial* [Str07] can be added to the system, and allows the contraction rule to be used in its atomic form only, while retaining completeness.

$$\mathsf{m}\,\dfrac{(A\wedge C)\vee(B\wedge D)}{(A\vee B)\wedge(C\vee D)}$$

The general contraction rule can be obtained from atomic contraction and this rule by duplicating all atoms in a given structure A, and then reshuffling these atoms to build the structure $A\vee A$.

Beyond the design of such advanced inference rules, the ideas developped in the calculus of structures have lead to the definition of an even more general setting. This formalism, called *open deduction* [GGP10], pushes further the collapse of the logical level with the deduction level as performed in the calculus of structures, by allowing connectives, and therefore inference rules, to apply on derivations. In this setting, the notion of branch is reintroduced by the fact that in a derivation $\mathscr{D}_1\wedge\mathscr{D}_2$, any inference rule applied only inside $\mathscr{D}_1$ can be considered in a different branch as a rule applied inside $\mathscr{D}_2$.

Example 3.8. *Below is shown a derivation in the system for classical logic in open deduction, where several rule instances are applied in parallel. The premise of this derivation is* B *and its conclusion is the structure* $((A\wedge A)\vee\top)\wedge B$ *that can be read off the conclusion of the several subderivations appearing under the* $\wedge$ *and* $\vee$ *connectives.*

$$\equiv\dfrac{\dfrac{B}{\top}}{\mathsf{i}\!\downarrow\dfrac{}{\mathsf{c}\!\uparrow\dfrac{A}{A\wedge A}\;\vee\;\mathsf{w}\!\uparrow\dfrac{\neg A}{\top}}\;\wedge\;B}$$

3.2 Nested Sequents

An alternative to the radical treatment of proof objets in the calculus of structures is offered by the formalism of *nested sequents*, where deep inference is implemented in a setting that retains a difference between the basic logical level of formulas and the meta-level of sequents. Nested sequents have been introduced in the study of deep inference calculi for modal logics [Brü10], but this notion was studied in various forms and under various names in earlier[18] work, in particular in the work of Kashima on tense logics [Kas94]. Since it will be used here as an alternative to the calculus of structures, we follow the work of Brünnler, with some divergence in the design of inference rules. As before, we illustrate the notions described by the representation of classical logic in this setting.

Syntactic structure. The basic layer of this formalism is also plain formulas, as described previously, and we use the same notational conventions. These formulas are used as part of a metal-level that is an enriched form of sequent: the most basic requirement is that a sequent can contain object containing sequent themselves, but the precise definition given for such a nested form of sequents depends on the logic represented. The idea is to represent at the meta-level all the connectives available in the logic, so that the toplevel connective of a formula can be decomposed into its metal-level equivalent, making its subformulas accessible. In classical logic, this requires the use of two kinds of sequents.

Definition 3.9. *A classical* nested sequent *is either a classical formula or a multiset of* branch-sequents*, and a branch-sequent is a multiset of nested sequents.*

We will denote nested sequents and branch-sequents by small letters such as δ, γ and ϕ where they are manipulated as objects, and they are represented with the syntaxes $[\vdash \delta, \cdots, \phi]$ and $[\delta; \cdots; \phi]$ respectively[19] — omitting the brackets when they are not necessary to the understanding. For the sake of clarity, we will denote a nested sequent containing only one formula A as this formula itself, and use the same notations Γ, Δ and so on for lists of branch-sequents $\delta_1, \cdots, \delta_n$ and of nested sequents $\gamma_1; \cdots; \gamma_n$, the nature of this list being clear from the context.

Example 3.10. *The notation* $\vdash \Gamma, \delta, [A; [\vdash \Delta, B \wedge C]]$ *describes a nested sequent that contains a list of branch-sequents* Γ*, a branch-sequent* δ *and another branch-sequent containing two nested sequents:* A *and a sequent containg* Δ *and the formula* $B \wedge C$*.*

In this definition, nested sequents correspond to traditional sequents, where the comma is interpreted as disjunction, and branch-sequents represent *» branchings «* separating several sequents, where the semicolon is interpreted as conjunction. The benefit of this approach is that the decomposition of connectives into the meta-level allows to keep track of the parts of the sequents that were already treated. This has the consequence that a nested sequent system can enjoy the subformula property in its usual form: the formulas in the premises of a rule instance are all subformulas of formulas in the conclusion of this instance.

[18] For a discussion of related work on nested sequents, see [Brü10].

[19] Notice that the definition would allow for *infinitely nested* sequents, but we will not use such infinite objects, and give no syntax to define them, since it is of no use in the representation of standard logics.

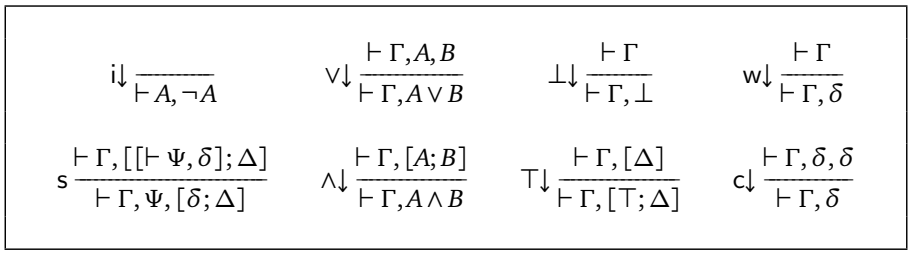


Figure 6: Inference rules for system KN

In order to define inference rules that can be applied on sequents nested inside other sequents, we need to define a notion of context similar to the one used in the calculus of structures. In the following we use the term *» sequent «* in a generic way for objects that are either nested sequents or branch-sequents.

Definition 3.11. *A* context *is a sequent containing a special sequent called* hole*, that can be replaced by any other sequent — a nested sequent or branch-sequent depending on the configuration.*

The formalism of nested sequents allows inference rules to have more than one premise, as used in the work on modal logics in this setting [Brü10]. However, we choose here to use rules with only one premise, because it replaces the branching mechanism with the distribution performed by the switch rule that we consider to be more elegant and more general. Notice that branching is compatible with the nested application of inference rules, so that one can have the following rule:

$$\wedge \frac{\xi\{A\} \qquad \xi\{B\}}{\xi\{A \wedge B\}}$$

in classical logic, but this might lead to complications in other logics, depending on the possible contents of the context ξ and on the position of the hole in $\xi\{-\}$.

The KN classical system. As an intermediate between LK and the KS system, we define the nested sequent system KN, for which the inference rules are given in Figure 6. It is based on classical nested sequents as defined previously, and has rules for units since there is no congruence as in the calculus of structures, and also has rules to decompose the connectives $\vee$ and $\wedge$ into the meta-level.

Although some rules are exactly the same as in the sequent calculus LK, others reflect the nested structure of proof in this system. The $\top\downarrow$ rule corresponds for example to the *» closing «* of a branch, while $\wedge\downarrow$ corresponds to the *» opening «* of a new branch, to which no hypotheses have been given yet. Then, the switch rule is similar to the switch[20] of the calculus of structures, simply moving hypotheses from one sequent to another, nested in the first one. Finally, notice that the weakening and contractions are almost the same as the sequent calculus, but they allow to erase and duplicate full sequents rather than just formulas.

[20]Rules in nested sequent systems usually apply only on sequents, either reduced to one formula or complete ones, but here it moves several sequents — this design was chosen to obtain a self-dual rule.

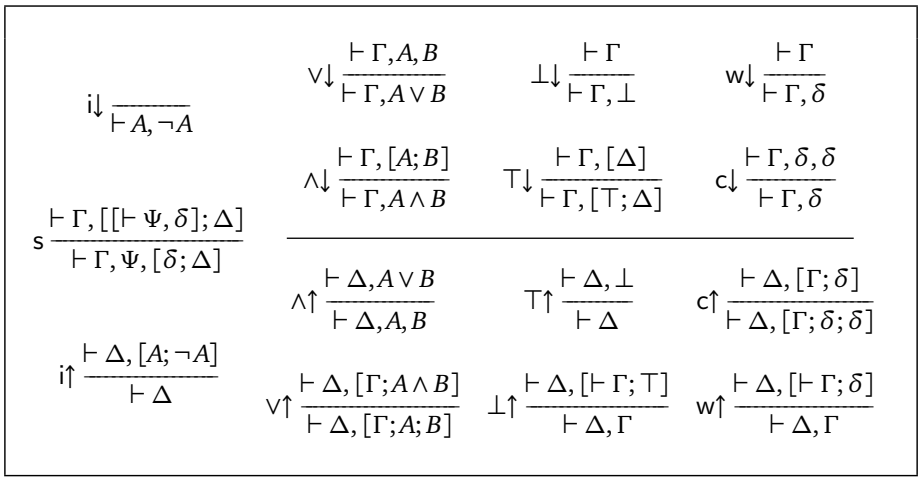


Figure 7: Inference rules for system SKN

The rules of this KN system are essentially the same as the rules for KS, but the distinction between the logical level and the meta-level of nested sequents and branch-sequents allows to have only rules respecting the subformula property in its usual acceptation. The cut rule, which can be defined as the dual of the identity rule can be added to the system and of course, it does not respect this property:

$$\mathsf{i}\uparrow \frac{\vdash \Delta, [A; \neg A]}{\vdash \Delta}$$

Example 3.12. *Below are shown two derivations in the* KN *system, where one can observe the additional steps required to deal with the meta-level in nested sequents, in comparison with the calculus of structures, and one derivation illustrating the use of the cut and its connection to identity.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash [B]}{\vdash [B; \top]}\top\downarrow}{\vdash B \wedge \top}\wedge\downarrow}{\vdash A, B \wedge \top}\mathsf{w}\downarrow}{\vdash A \vee (B \wedge \top)}\vee\downarrow \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdash \Gamma, [\vdash \Delta; \top], [\vdash \Delta, \bot]}{\vdash \Gamma, [\vdash \Delta, \bot; \top], [\vdash \Delta, \bot]}\bot\downarrow}{\vdash \Gamma, [\vdash \Delta, \bot; \top], [\vdash \Delta, \bot; \top]}\top\downarrow}{\vdash \Gamma, [\vdash \Delta, \bot; \top]}\mathsf{c}\downarrow}{\vdash \Gamma, [\vdash \Delta, \bot; \top; \vdash A, \top]}\mathsf{w}\downarrow \qquad \dfrac{\dfrac{\dfrac{\vdash \Gamma, [A]}{\vdash \Gamma, [\vdash A, \neg A; A]}\mathsf{i}\downarrow}{\vdash \Gamma, A, [\neg A; A]}\mathsf{s}}{\vdash \Gamma, A}\mathsf{i}\uparrow$$

Symmetry in nested sequents. The notions of duality and symmetry, as they were described in the setting of the calculus of structures, can also be considered in nested sequents. The dualisation of a sequent is derived from duality of connectives and correspondences between meta-level objects and logical constructions, so that the dual of a nested sequent is a branch-sequent, and so on. Then, the dualisation of any rule can be defined as in KS, by reversing the rule and dualising its sequents, as was done for example with identity and cut. This leads to the symmetric SKN system, defined by the set of inference rules given in Figure 5.

This system follows the same scheme as the symmetric generalisation SKS of the KS system, and it also has one rule, the switch rule, which belongs to both up and down fragments — because this rule is self-dual. Notice that the up rules seem slightly more complex than the down rules, because we have chosen to present all rules as applied on nested sequents, and not branch-sequents.

Remark 3.13. *The system* SKN*, as well as* KN*, can be divided into several fragments with different purposes, as done in the calculus of structures. The identity fragment* $\{\mathsf{i}\downarrow, \mathsf{i}\uparrow\}$ *deals with the identity of formulas and the structural fragment* $\{\mathsf{w}\downarrow, \mathsf{c}\downarrow, \mathsf{w}\uparrow, \mathsf{c}\uparrow\}$ *with the erasure and duplication of sequents. Then the* s *rule is particular and could form alone the* meta-logical *fragment, dealing with logical connections at the level of sequents, while the logical fragment* $\{\wedge\downarrow, \vee\downarrow, \top\downarrow, \bot\downarrow, \wedge\uparrow, \vee\uparrow, \top\uparrow, \bot\uparrow\}$ *deals with the decomposition and recomposition of connectives.*

The introduction of dual rules in a nested sequents system has the same use as in the calculus of structures, but some dual rules are more interesting than others. For example, the cut rule is crucial as it corresponds to the cut rule in the sequent calculus, and the dual structural rules can be useful in the study of the dynamics of proofs, and computational interpretations. However, the duals of logical rules simply express the perfect duality between $\wedge$ and $\vee$, and their units, in the classical setting. This is why $\wedge\uparrow$ is only *reversing* the effect of $\vee\downarrow$, and the other way around, so that they seem not to be of any use to proof construction for example[21].

Example 3.14. *Below are shown derivations in the symmetric* SKN *system, where the particularities of the up fragment are illustrated. In the first one, shown on the left, one can observe that the* $\bot\uparrow$ *rule is not only dual to* $\bot\downarrow$ *but also » opposite « to the rule* $\top\downarrow$ *in the sense that they perform an operation in different directions. On the right, the derivation shows how up rules can be used to build formulas that will match a given hypothesis, much in the style of natural deduction.*

$$\dfrac{\dfrac{\dfrac{\vdash [\vdash \Gamma, A]}{\vdash [\vdash \Gamma, A; \top]}\,\top\downarrow}{\vdash \Gamma, [A; \top]}\,\mathsf{s}}{\vdash \Gamma, A}\,\bot\uparrow \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \Gamma, [A]}{\vdash \Gamma, [A; \vdash B \vee \neg A, \neg B \wedge A]]}\,\mathsf{i}\downarrow}{\vdash \Gamma, [A; \vdash B \vee \neg A, \neg B \wedge A]]}\,\mathsf{s}}{\vdash \Gamma, \neg B \wedge A, [A; \vdash B \vee \neg A]]}\,\wedge\uparrow}{\vdash \Gamma, \neg B \wedge A, [A; \vdash B, \neg A]]}\,\mathsf{s}}{\vdash \Gamma, \neg B \wedge A, B, [A; \neg A]}\,\mathsf{i}\uparrow}{\vdash \Gamma, \neg B \wedge A, B}$$

Finally, the derivation below shows how the rules of the up fragment can introduce some redundancy in the meta-level, inducing the use of additional switches.

$$\dfrac{\dfrac{\dfrac{\vdash \Gamma, [\vdash A, \Delta]}{\vdash \Gamma, A, [\vdash \Delta]}\,\mathsf{s}}{\vdash \Gamma, A, [\vdash \Delta; \top]}\,\top\downarrow}{\vdash \Gamma, A, \Delta}\,\mathsf{w}\uparrow$$

[21] Notice however that the KN system being complete, the dual logical rules are admissible in this system, and from this we can conclude that all logical rules are invertible.

3.3 Logical Flow and Permutations

In the theory of nested proof systems, the structure of proofs and the interaction of inference rule instances are even more important than in natural deduction or in the sequent calculus. Indeed, the ability to modifiy the contents of nested sequents and structures by applying inference rules generates new interactions between rule instances, and in particular creates many situations of possible permutations, often trivial but sometimes more complicated than in shallow formalisms. It is therefore important to adapt the tools developped in the traditional setting, as we will often rely on relations between instances, and permutations, in the study of nested proof transformations.

Nested logical flow. The foundamental tool of our approach to the structure of proofs, the flow-graphs, was designed in shallow formalisms to observe the effect of inference rule instances on occurrences of formulas within a proof. However, in nested proof systems, not only formulas are manipulated but also nested sequents and structures. The situation is different in the two formalisms we presented:

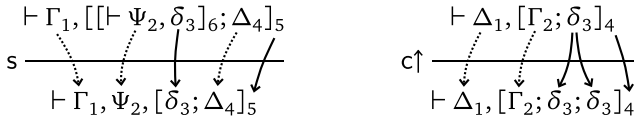
- In nested sequents, the tools developped to study formulas in shallow systems are still valid, but we might need to extend them to handle the objects, nested sequents themselves, since they can be moved and manipulated as well.
- In the calculus of structures, inference rules often decompose and recompose structures, so that we need to keep track of how a structure can be modified by adapting the notions of particle and father.

In the setting of nested sequents, all definitions given previously for the notions of particle, father and flow-graph are valid, and we can simply extend the definition of particles to extend the whole toolset.

Definition 3.15. *In a nested sequent system, a* particle $\underline{A}$ *or* $\underline{\delta}$ *of a derivation* $\mathscr{D}$ *is an occurrence of a formula or subformula* A*, or of a sequent* δ*, that appears in a sequent within* $\mathscr{D}$*.*

The extension of the flow-graphs to handle sequent particles assumes that any rule instance, in nested sequents, contains the information of which sequents in the conclusion and premises correspond, through the use of indexes. This is usally clear from the scheme given for an inference rule, and can be explicitly described when needed — it is part of the definition of a system.

Example 3.16. *Here are two inference rules of the classical symmetric system* SKN*, annotated with indexes on sequents, where the connections induced by the definition of flow-graphs on sequents are illustrated with arrows.*



Note that we use a simplified notation where one index is given for a bunch of sequents Γ *with the purpose of denoting the several indexes used on sequents with* Γ*, which have the expected indexes in the premise and conclusion.*

In the calculus of structures, the situation is more complicated. Indeed, in the shallow formalisms, it often happens that formulas are composed or decomposed, as in introduction and elimination rules in natural deduction, but a formula is never *modified* in the sense that we would identify a formula containing different subformulas, through indexes. If we consider for example the following instances in LJ and KS respectively:

$$\rightarrow_{\mathsf{e}} \frac{\Gamma_1 \vdash A_3 \quad \Gamma_1 \vdash (A_4 \rightarrow B_2)_5}{\Gamma_1 \vdash B_2} \qquad\qquad \mathsf{s}\frac{(A_1 \vee B_2) \wedge C_2}{A_1 \vee (B_2 \wedge C_3)}$$

we can see that it would be problematic to apply the same policy of indexing to the switch rule as to natural deduction rules, since in the perspective of deep inference, A is *» moved inside the conjunction $B \wedge C$ «* and thus the conjunction structures in the premise and conclusion should be labelled with the same index. Moreover, the use of a congruence complicates the assignment of indexes. First, we can transfer directly the notion of particle by considering structures instead of formulas.

Definition 3.17. *In a system in the calculus of structures, a* particle $\underline{A}$ *of a derivation $\mathscr{D}$ is an occurrence of a structure A that appears in any context within $\mathscr{D}$.*

Then, the key to the definition of a notion of flow-graph adequate to the setting of the calculus of structures is to manage the way indexes are assigned in a correct way. In order to clarify the way inference rules should handle indexes following the intuition, can state the policy for indexing in the calculus of structures as follows:

(i) Only representations of structures using a minimal amount of symbols in their writing are considered for indexing.

(ii) A particle in a structure is assigned an index if it appears in any of the considered representation of this structure.

(iii) If two particles with the same main connective have particles of same indexes on one side this main connective, then they have the same index as well.

where the *main connective* of a structure is simply the one[22] that appears at toplevel in the syntax tree of the formulas represented by this structure. In the KS system, the meaning of the first rule is that we will not consider the units that can be added or removed by the congruence, and always use representations in the normal form where negation has been pushed to the atoms. The second rule says in KS that in a structure such as $A_1 \wedge B_2 \wedge C_3$, we assign indexes to both particles $(A_1 \wedge B_2)_4$ and $(B_2 \wedge C_3)_5$ and thus allow to handle cases where any of them is used. Finally, the third rule identifies the recomposition of structures, so that the assigment obtained for the switch in KS is the following:

$$\mathsf{s}\frac{((A_1 \vee B_2)_5 \wedge C_2)_4}{(A_1 \vee (B_2 \wedge C_3)_4)_5}$$

[22] The toplevel connective can normally not be changed by an equation from the congruence, but if it does then two particles should have corresponding sets of possible toplevel connectives to match.

and corresponds to the intuition that the disjunction $A_1 \vee -$ is moved towards the B_2 inside the conjunction. With such rules to define how indexes are assigned in an inference rule instance to particles, the definitions given for the notions of father and flow-graph can be transferred into the calculus of structures in a satisfying way.

Example 3.18. *Here is a derivation in the* SKS *symmetric system for classical logic, where structures are annotated with indexes to indicate how the flow-graph for this derivation can be built. Not all indexes are indicated, but enough for the flow-graph to be read off the derivation.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{b_3}{(b_3 \wedge (a_1 \vee \overline{a}_5)_9)_7}\,\mathsf{i}\downarrow}{(a_1 \vee (b_3 \wedge \overline{a}_5)_7)_9}\,\mathsf{s}}{(a_1 \vee (b_3 \wedge b_3 \wedge \overline{a}_5)_7)_9}\,\mathsf{c}\uparrow}{(a_1 \vee (((\overline{a}_2 \vee a_4)_8 \wedge b_3)_6 \wedge b_3 \wedge \overline{a}_5)_7)_9}\,\mathsf{i}\downarrow}{(a_1 \vee (((\overline{a}_2 \wedge b_3)_6 \vee a_4)_8 \wedge b_3 \wedge \overline{a}_5)_7)_9}\,\mathsf{s}}{(a_1 \vee ((\overline{a}_2 \wedge b_3)_6 \vee (a_4 \wedge b_3 \wedge \overline{a}_5)_7)_8)_9}\,\mathsf{s}$$

Active and passive sequents and structures. The principal application of the flow-graph tool that we used in standard intuitionistic systems was the definition of active and passive formulas in rule instances. In a nested proof system, this is also an essential element, that we need to adapt. One of the benefits of nested sequents is as usual that the definitions given in standard formalisms can be applied directly, so that particles are considered active or passive in a rule instance in this setting under the same conditions — and considering that the status of a sequent is either *» nested sequent «* or *» branch-sequent «*. As a consequence, the notion of multiplicity is directly imported from the standard setting.

In the calculus of structures, once again, the situation is more complicated, because there is no notion of *status*, as in a formula that would be made accessible by decomposing another formula. The characteristic point in any structure active in a rule instance, outside being erased or duplicated, is that its contents have been modified: not only reorganised, but extended or diminished by the introduction or escape of a substructure. This is expressed in the following definition.

Definition 3.19. *In a rule instance* r *in the calculus of structures, a particle* $\underline{A}$ *is said to be* passive *if and only if it is the father or child of exactly one particle* $\underline{B}$*, and there is no particle in* r *that appears inside of* $\underline{A}$ *and outside of* $\underline{B}$*, or outside of* $\underline{A}$ *and inside of* $\underline{B}$ *— and* $\underline{A}$ *is said to be* active *in any other case.*

Example 3.20. *The two instances of inference rules of the* SKS *system shown below illustrate the definition of active particles, by using the notation* $[i]$ *for the index of an active particle. On the left, all duplicated particles are active, while on the right the moved particles are passive.*

$$\mathsf{c}\downarrow \dfrac{(((a_{[1]} \wedge b_{[2]})_{[4]} \vee (a_1 \wedge b_2)_4)_{[6]} \vee C_3)_5}{((a_{[1]} \wedge b_{[2]})_{[4]} \vee C_3)_5} \qquad \mathsf{s}\dfrac{((A_1 \vee B_2)_{[5]} \wedge C_3)_{[4]}}{(A_1 \vee (B_2 \wedge C_3)_{[4]})_{[5]}}$$

As in standard formalisms, active particles represent structures that have been genuinely affected by the application of an inference rule on a structure. From this we can import the definition of the multiplicity of a rule instance in the calculus of structures, from the standard setting.

Permutations of rule instances. The status of particles in a given rule instance, describing which particles are involved in an inference step and which are not, is an important observation because it provides information on the possible interaction between rule instances, and in particular their permutation. In particular, the local nature of the rewriting performed by one rule instance on a structure allows to make the same crucial observation on permutations than in shallow formalisms.

Proposition 3.21. *In any derivation $\mathscr{D}$, if two rule instances r_1 and r_2 appearing one above the other share no active particle, then they can be trivially permuted.*

This observation holds for the same reasons as in shallow formalisms such as the sequent calculus: since a passive particle appears exactly once in the premise and once in the conclusion, and is not essentially modified, any rewriting performed on this particule above the premise could also be done below the conlusion.

The cases of trivial permutation in nested formalisms are quite important, in the sense that they appear in any permutative transormation, since instances are usually considered in some unspecified context, and a rule instance is therefore an object that can always trivially permute with the other instances of its context. The following permutation:

$$\mathsf{r}_2\dfrac{\mathsf{r}_1\dfrac{\xi\{C\}\{D\}}{\xi\{A\}\{D\}}}{\xi\{A\}\{B\}} \qquad \longleftrightarrow \qquad \mathsf{r}_1\dfrac{\mathsf{r}_2\dfrac{\xi\{C\}\{D\}}{\xi\{C\}\{B\}}}{\xi\{A\}\{B\}}$$

where $\xi\{-\}\{-\}$ is some context with two separate holes, can usually be implicitly treated in the description of permutative transformation because it preserves all simple properties that one can reasonably define on nested rules and derivations — except the height of the derivation above the instances r_1 and r_2, of course. This case of permutation corresponds to permutations in the sequent calculus that are either implicit — if the two instances are located in different branches — or are also trivial, for example when two formulas in a sequent are decomposed without branching, so that they can be decomposed in any order.

Then, another case of trivial permutation is specific to the nested setting, as it happens when an instance superficially modifies a sequent or structure and another modifies some part of it, nested deep inside. The permutation:

$$\mathsf{s}\dfrac{\mathsf{r}\dfrac{\xi\{(B\vee C)\wedge D\}}{\xi\{B\vee(C\wedge D)\}}}{\xi\{A\vee(C\wedge D)\}} \qquad \longleftrightarrow \qquad \mathsf{r}\dfrac{\mathsf{s}\dfrac{\xi\{(B\vee C)\wedge D\}}{\xi\{(A\vee C)\wedge D\}}}{\xi\{A\vee(C\wedge D)\}}$$

in KS is a striking example, where the r instance *flows through* the switch instance just as A is moved through the conjunction. In general, permutations in nested systems, when they are not trivial, can induce complex situations and require highly technical treatments [GS11a].

3.4 Normal Forms in Nested Proof Systems

The structure of proof objects, in the setting of nested sytems, is in general much more complex than the structure of shallow proofs as described by formalisms such as natural deduction or the sequent calculus. As we have seen, the use of structural rules, even in natural deduction, or of left rules as in the sequent calculus, induces the possibility of permuting rule instances in a given proof and raises the question of possible normal forms with regards to these permutations. In nested formalisms, the collapse of branching and the ability to apply inference rules inside formulas or in nested sequents creates many new cases of permutations.

In the sequent calculus, it is possible to permute some rule instances to try to have them only at particular positions in a proof. This can be mimicked in nested sequents systems, but it often makes less sense than in formalisms using a two-level presentation of logic and deduction. Moreover, the permutation of rule instances might lead to the complete erasure of some instances from a given proof. In order to understand what kind of normal forms can be useful, one can consider the two following categories:

- A normal form can be defined through restrictions on the possible locations of instances of an inference rule in a proof, by stating that any proof should be divided into portions, each using only a subset of inference rules, or defining in which context an inference rule can be applied.

- A normal form can be defined through a restriction on the set of inference rules used in a proof system, by stating for example that a rule of this system is admissible for the other rules, and that proofs not using it are considered as the normal ones.

In each case, the important point is that the restrictions imposed on proof does not break the completeness of the system. The goal of a normal form is indeed to define some subset of the set of all possible proofs in a system, which is as small as possible, or respects some good properties, but is *representative* in the sense that it is logically complete with respect to the unrestricted system. The most prominent examples of these two categories of normal forms are probably, in the framework of the sequent calculus, focused proofs [And92] and cut-free proofs. In the setting of deep inference, these categories are illustrated by several normal forms, defined in various systems. Since the two formalisms of the calculus of structures and nested sequents are similar on this issue, and the question has been studied mainly in the former, we will only describe normal forms in the calculus of structures.

Decomposition. The ability to apply an inference rule deep inside a structure allows for more permutations in the calculus of structures as in shallow formalisms, so that normal forms based on the separation of different fragments of a system, within proofs, can be defined rather easily in comparison to the shallow setting. In the classical system KS for example, where contraction can be reduced to its atomic form by introducing the medial rule m, it is possible to freely permute down atomic contraction instances [Brü06c].

In this system, given a proof $\mathscr{P}$ of some structure A, it is possible to transform it, by permutations only, into a proof $\mathscr{P}'$ where instances of the atomic contraction $\mathsf{ac}\downarrow$ are all located at the bottom:

$$\begin{array}{r} \mathsf{KS}\Vdash \\ A \end{array} \quad \longrightarrow \quad \begin{array}{r} \mathsf{KS}\backslash\mathsf{ac}\downarrow\Vdash \\ B \\ \{\mathsf{ac}\downarrow\}\Vdash \\ A \end{array}$$

Several *decomposition theorems* can be proved in the calculus of structures, and some have been stated in systems for classical logic [Brü03] and systems for linear logic [Str03a], but this is in general a natural question in many systems. In a given system S with a set of inference rules $\{\mathsf{r}_1, \cdots, \mathsf{r}_n\}$, one can for example define three subsets of rules $\mathscr{R}_1 = \{\mathsf{r}_1, \cdots, \mathsf{r}_i\}$, $\mathscr{R}_2 = \{\mathsf{r}_{i+1}, \cdots, \mathsf{r}_j\}$ and $\mathscr{R}_3 = \{\mathsf{r}_{j+1}, \cdots, \mathsf{r}_n\}$, and possibly show the completeness of the set of proofs in S which have the following shape:

$$\begin{array}{r} \mathscr{R}_3\Vdash \\ C \\ \mathscr{R}_2\Vdash \\ B \\ \mathscr{R}_1\Vdash \\ A \end{array}$$

For example, in the variant of the classical system SKS where the medial m is introduced and atomic rules $\mathsf{ai}\downarrow$, $\mathsf{aw}\downarrow$ and $\mathsf{ac}\downarrow$ are used instead of their non-atomic versions — and the same in the up fragment —, almost all rules can be separated from the others [Brü03]. The subsets of rules used in this decomposition theorem are $\{\mathsf{ac}\downarrow\}$, $\{\mathsf{aw}\downarrow\}$, $\{\mathsf{ai}\uparrow\}$, $\{\mathsf{s}, \mathsf{m}\}$, $\{\mathsf{ai}\downarrow\}$, $\{\mathsf{aw}\uparrow\}$ and $\{\mathsf{ac}\uparrow\}$, used from bottom to top. A similar result has been obtained in linear logic without additives [GS11a].

Cut elimination. On the side of normal forms defined by the restriction on the set of rules used in proofs, cut elimination is the most important transformation, which has been studied in detail in the calculus of structures. In classical logic, the proof [Brü03] by Brünnler for $\mathsf{KS} \cup \{\mathsf{i}\uparrow\}$ is surprisingly simple. It is based on an idea similar to the substitutive method used for detour elimination in the intuitionistic natural deduction system NJ. It relies on several particularities of the KS system. First, the cut rule can be made atomic, since compound formulas can be built from atoms by switch instances in the configuration that would result from a general cut. Then, it uses the fact that cuts can be restricted to apply only at the outer level of structures, since switches can be used to move the cut structures inside the context where the cut would have been used. The first step is the extraction of two objects from one given proof:

$$\begin{array}{r} \mathscr{P}_1\Vdash \\ B \vee a \end{array} \quad \longleftarrow \quad \mathsf{ai}\uparrow \dfrac{\begin{array}{r} \mathscr{P}\Vdash \\ B \vee (a \wedge \overline{a}) \end{array}}{B} \quad \longrightarrow \quad \begin{array}{r} \mathscr{P}_2\Vdash \\ B \vee \overline{a} \end{array}$$

where the objects $\mathscr{P}_1$ and $\mathscr{P}_2$ are obtained by removing $\overline{a}$ and a from the proof $\mathscr{P}$, respectively. However, the object $\mathscr{P}_1$ is actually not a proof: it is broken in some identity instances, the ones involving the atom a. The second step is to fix $\mathscr{P}_1$ by substituting copies of $\mathscr{P}_2$ in place of these broken identity instances, and replace a by B throughout $\mathscr{P}_1$. Finally, using a contraction, the resulting proof of $B \vee B$ is turned into a proof of B where the considered cut has been removed. This method can be used immediatly in the symmetric SKS system, because the cut rule $\mathsf{ai}{\uparrow}$ can simulate $\mathsf{i}{\uparrow}$ which allows to encode all other rules of the up fragment, using identity and switch:

$$\mathsf{r}{\uparrow}\,\dfrac{\xi\{A\}}{\xi\{B\}} \quad\longrightarrow\quad \begin{array}{c} \mathsf{i}{\downarrow}\,\dfrac{\xi\{A\}}{\xi\{A \wedge (B \vee \neg B)\}} \\[2ex] \mathsf{s}\,\dfrac{\xi\{A \wedge (B \vee \neg B)\}}{\xi\{B \vee (A \wedge \neg B)\}} \\[2ex] \mathsf{r}{\downarrow}\,\dfrac{\xi\{B \vee (A \wedge \neg B)\}}{\xi\{B \vee (A \wedge \neg A)\}} \\[2ex] \mathsf{i}{\uparrow}\,\dfrac{\xi\{B \vee (A \wedge \neg A)\}}{\xi\{B\}} \end{array}$$

In the system LS for linear logic, the situation is more complicated and another technique was developped to prove the cut elimination result internally. It is based on the *splitting* lemma [GS09], which states that interleaved parts of a proof can be extracted, in such a way that a transformation similar to the one used in KS, involving the plugging of proofs into another one, is made possible. Notice that no proof of cut elimination based on permutations has been proposed in these systems, because the complex interactions between rule instances in a nested setting makes it highly difficult to design a procedure for which a measure can be found.

Symmetric normalisation. One particular feature of the calculus of structures is the natural approach it has to open derivations, which are actually the important objects in this theory, rather than complete proofs — a proof is not a symmetric object, while symmetry is the most important principle underlying the techniques developped in this setting. However, the traditional view of cut elimination is the possibility of removing all cuts from a proof, but this is only possible for complete proofs. In the nested formalisms, it is possible to define a generalisation of the cut elimination result which is centered on derivations, and where the use of the whole up fragment, rather than just the cut rule, is crucial. For example, in the SKS system, the following transformation can be performed [Brü06b]:

$$\begin{array}{r} A \\ \mathsf{SKS}\,\| \\ B \end{array} \quad\longrightarrow\quad \begin{array}{r} A \\ \mathsf{KS}{\uparrow}\,\| \\ C \\ \mathsf{KS}{\downarrow}\,\| \\ B \end{array}$$

where $\mathsf{KS}{\downarrow}$ and $\mathsf{KS}{\uparrow}$ are respectively the down and up fragment of the SKS system, and this is indeed a generalisation of cut elimination in the sense that, if the original derivation is a proof, then its premise is $\top$ and because of the shape of up rules, C must be logically equivalent to $\top$, so that we have in the end a cut-free proof of B, in the KS system. This notion of normal form should be considered as the one to use in a symmetric formalism such as the calculus of structures, and it is made possible by the dualisation of all inference rules from the down fragment.

Chapter 2

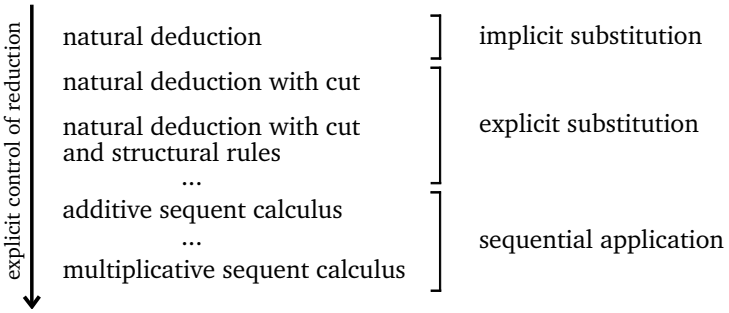
Logical Foundations for Computation

This chapter is concerned with the use of logical systems, and in particular proof systems as manipulated in structural proof theory, as a cornerstone in the design of sound principles for computation. Indeed, the theoretical modelisation of any computational device is a higly complex task where any subtle error can have many consequences, especially if this model is *Turing-complete*. In the following chapters, the two main approaches to logical foundations for computation are considered:

- In the » *proofs-as-programs* « paradigm, a correspondence is established from the syntax between the proofs of a given logical system and the programs of a computational model — usually based on rewriting —, while at the dynamic level, a correspondence is established between a process of normalisation of proof objects and the execution mechanism defined for programs.
- In the » *proof-search-as-computation* « paradigm, a more abstract viewpoint is adopted, where a formula is seen as declarative program describing a task, while the process of building a proof of this formula in a given logical system is identified as the computation that eventually produces the result described by the formula.

Following the first methodology, the most prominent example of the connection between proofs and programs is the *Curry-Howard correspondence*, which relates proofs in intuitionistic natural deduction and functional programs as described in the λ-calculus. We recall here the construction of this correspondence, through the use of different logical systems as type systems for variants of the λ-calculus. Based on the description of systems for intuitionistic logic given in Chapter 1, we show how the refinement of the transformation used for proof normalisation induces a refinement of the operational mechanisms associated with the λ-calculus, from the meta-level specification of substitution to the fine-grained propagation of explicit substitutions. The two sides of the correspondence are emphasised by a separation between the study of the properties of untyped λ-calculi and the results obtained by restricting the set of considered terms to well-typed ones.

Beyond the original description of the simply typed λ-terms as a well-behaved subset of all λ-terms, the correspondence between intuitionistic proofs and various λ-calculi has been refined in two steps. First, introducing a cut rule to perform the normalisation of proofs by local rewriting is equivalent to the extension of the pure λ-calculus with an internal notion of substitution, with a dedicated syntax inside the language, called *explicit substitution*. Then, decomposing the inference rules of natural deduction to have separate structural rules leads to a more precise handling of the erasures and duplications required by the substitution process. When rules are designed following the introduction/elimination scheme of natural deduction, the correspondence involves a *standard* variant of the λ-calculus. The figure below illustrates the different steps of explicitation in intuitionistic systems and λ-calculi.



The second step of refinement, shown in the bottom part of the figure, is the use of the sequent calculus, where inference rules are designed following the left/right scheme and the correspondence can be established with λ-calculi using a *sequential* form of application. This correspondence has been less studied than its equivalent in natural deduction, and there is no » *standard* « among all of the different calculi suggested to represent sequent proofs. We present one correspondence and discuss the operational properties of what we call calculi with *pure explicit substitutions*. As before, the use of structural rules induces refined calculi.

Following the second methodology, the development of programming languages where formulas encode the programs has been most often based on the *resolution* technique, which does not induce the same theoretical clarity than the approaches based on structural proof theory. After the introduction of *uniform proofs* as a mean of designing logic programming languages in a rigorous way, the definition of the *focusing* normal form in linear logic has been instrumental in the development of the approach based on the sequent calculus. We recall in this chapter the basics of the sequent calculus description of linear logic, and its impact on the studies of the computational aspects of logic, and then describe the use of proof search as a highly abstract programming language. Finally, we also recall the definition of the focusing technique for linear logic, that will be the starting point of a generalisation of normal forms following the principles of polarities in another chapter. It should also be noticed that proof search is more general than functional computation since it allows not only to compute but also to *reason* about computation, as for example in the setting of LF and its variants.

1 Proof Normalisation as Functional Computation

The development of the λ-calculus as a computational model, which offers a much more » *mathematical* « approach to computer programs than other models closer to real-world machines, has lead to the development of a variety of new programming languages, such as those from the family of Lisp [McC60], where the primary tool for writing programs was to define functions and apply them to functions, rather than update values in registers and perform tests and loops. However, while some λ-terms may be well-behaved, there is no syntactic distinction *a priori* between a program which computes a result, and a program that will loop forever and never give back any result. A safe approach to programming would thus require to define a subset of λ-terms that are proved to be well-behaved, thus returning a result after a finite time, and use only those terms as programs.

An important step in the development of this *functional programming paradigm* was the introduction of *type systems*, allowing to ensure that a functional program terminates and describing the programs it can safely interact with. On a theoretical level, this approach originates in the observation by Howard, in 1969[1], that λ-terms are isomorphic to intuitionistic proofs in natural deduction [How80], and that the reduction of λ-terms corresponds to the normalisation of intuitionistic proofs. This is the basic principle underlying the *typed* functional programming paradigm: only a subset of all possible λ-terms can be used as valid programs, but these terms are exactly the ones in correspondence with valid proofs. Then, a normalisation result obtained on the logical side ensures the termination of β-reduction, seen as a proof transformation. Moreover, each valid term is assigned a type, which is actually a logical formula — in the case of intuitionistic logic with only implication, this is called a *simple type* — and can be used to determine how the term can be used as a function or as an argument to another function. The original setting of the simply typed λ-calculus involves the following correspondences:

Logic	**Computation**
intuitionistic formula A	simple type A
proof $\mathscr{P}$ of A in NJ	closed λ-term t of type A
detour on B in $\mathscr{P}$	β-redex $(\lambda x.u)\,v$ in t, with v of type B
detour elimination in $\mathscr{P}$	strong β-reduction of t

and this particular correspondence, restricted to simple terms and to intuitionistic proofs in natural deduction, is called the *Curry-Howard correspondence*. We present now in more details this correspondence, and describe the basic type system, that can be extended to handle larger subsets of λ-terms or extensions of the λ-calculus.

[1] The original manuscript of Howard dates from 1969, but this was only published in 1980, and this was based on an earlier observation by Curry concerning the link between combinators and the axioms of intuitionistic logic [Cur34].

1.1 Natural Deduction and Typed λ-terms

In order to establish a correspondence between intuitionistic natural deduction and the λ-calculus, we need to consider types for λ-terms. Such a type is just a logical formula, and in the most basic case it can only be either an atom or an implication, in which case it is called a simple type. The intended interpretation of simple types is the following:

- An atom a is the type of a constant[2], or of a variable on which no information is available, so that it is not known if it is used as function or argument — it is possible to use only one atom, which is traditionally denoted by σ.
- An implication $A \to B$ is the type of a function which takes an argument of type A and produces a result of type B, and assigning such a type to a function means that it is considered » *unsafe* « to use it on an argument of another type than A, in the sense that it might not produce the expected result.

The idea there is to provide some information on the behaviour of a functional program, seen as a λ-term. For example, any term of type $A \to B \to A$ is a function of two arguments returning a result of the same type as its first argument — and if A and B have nothing in common, it is likely that this function » *forgets* « about its second argument and returns a result based on its first argument only, as the term $\lambda x.\lambda y.x$ for example. In order to extract this information from a given λ-term, it is necessary to inspect the syntactic structure of this term: this is the purpose of type systems, to validate or build a judgement stating that some λ-term can be assigned a certain type, under a set of assumptions on the type of its variables.

Definition 1.1. *A* typing judgement *is denoted by* $\Gamma \vdash t : A$ *and defined as the triple formed of a multiset* Γ *of typing assumptions, a* λ*-term* t *and a simple type* A*.*

In this definition, a *typing assumption* is the pair of a variable x and any type B, which is denoted by $x : B$ and means that we assume x to be of type B. The syntax for judgements is derived from the syntax of sequents, and multisets of assumptions will therefore be denoted by letters such as Γ, Δ or Φ. A type system is a way of defining a procedure to inspect the structure of a term and establish its behaviour as a certain type. This is usually done by providing rules of the shape of inference rules, each of them asserting that a term using a certain construct at toplevel can be assigned a type under the hypothesis that its subterms can be typed, as expressed in the premises of the rule, so that inspection is done inductively.

Note that there are two possible uses for a type system: either we want to find out what type can be assigned to a given term, and this is *type inference* — a key element in programming languages where type annotations are not required, such as ML —, or we want to verify that a term fits a given type, and this is *type checking*. Although the standard presentation is neutral with regard to this question, this can be emphasized within the rules, as done in *bi-directional typing* [PT98].

[2] If we add a set of constants to the λ-calculus, which is not so useful on a theoretical level but does correspond to a normal practice in implementations; for example, one can add infinitely many constants $1, 2, \ldots$ and given them all the type *int*, to have a native support for integers in the calculus, although this would not allow to have operations on integers *inside* the λ-calculus.

$$\texttt{var}\ \frac{}{\Gamma, x:A \vdash x:A} \qquad \texttt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t : A \to B}$$

$$\texttt{con}\ \frac{(\ c \text{ constant of type } A\ )}{\Gamma \vdash c : A} \qquad \texttt{app}\ \frac{\Gamma \vdash u:A \quad \Gamma \vdash t : A \to B}{\Gamma \vdash t\ u : B}$$

Figure 1: Type system S for the λ-calculus with constants

Simple types. The most basic type system for the λ-calculus is based on simple types, using only atoms and the implication, and it can establish that a given term can be assigned a certain type, while ensuring that applications within the term are valid, in the sense that they respect the types of the subterms used as function and argument respectively. This system, that we will call here S, is defined by the typing rules shown in Figure 1, and it includes a rule for constants, for the sake of clarity, but this rule is not required if we only use pure λ-terms. This defines a subset of all λ-terms called the *simply typed λ-calculus*, which is the canonical representant of typed calculi used in the literature [Rey98], although more elaborate systems are used in practice. The rules can be described as follows:

- The `var` rule says that under a set of assumptions including the fact that the variable x has type A, one can assign to x the type A.
- The `con` rule says that under any set of assumptions, a constant c of type A can be assigned type A.
- The `lam` rule says that under any set of assumptions Γ, one can assign to the term $\lambda x.t$ the type $A \to B$, provided that t can be assigned the type B under the set of assumptions $\Gamma \cup \{x:A\}$.
- the `app` rule says that under a set of assumptions Γ, the application $t\ u$ can be assigned the term B, provided that u can be assigned some type A, and the term t can be typed as a function of type $A \to B$.

As the shape of its rules suggests, the S system is used the same way as a logical proof system is used for proof search, except that the *inference* process is guided by the term to be typed — it is *syntax-directed* because there is always only one rule to apply, depending on the given term only. When it is used for type-checking, the situation is even simpler as it is just a matter of verifying that terms and types match through the rules. Following the similarity to proof systems, we call a *typing derivation* an object build from the rules of the system S, and the equivalent of a proof would be a *closed* typing derivation, where no subterm is left untyped. Thus, a typing judgement $\Gamma \vdash t : A$ is derivable in S when one can build a closed typing derivation with this judgement as a conclusion.

Definition 1.2. *A λ-term t is said to have type A when $\vdash t : A$ is derivable in* $\mathtt{S}$.

When a typing judgement $\Gamma \vdash t : A$ is derivable, we simply write $\Gamma \vdash t : A$ in $\mathtt{S}$, and the fact that t has type A in $\mathtt{S}$ can thus be written $\vdash t : A$. Now, there are several important properties of the $\mathtt{S}$ system to describe. The first one is related to its use as a type inference engine, which is given a term t and produces some type if t is a valid simply typed term, and fails if t is outside of this subset. In order to use this, type inference must terminate, either returning a type or through a failure, as can be proved by structural induction on a given term.

Proposition 1.3. *The typing process in* $\mathtt{S}$ *is terminating.*

As mentioned in the proof of termination, the typing process relies on the ability to » *rename* « atoms in a formula, by substituting all occurrences of an atom a by occurrences of another atom b in the formula. Indeed, a unique λ-term can have several types[3]. For example, the identity term $\lambda x.x$ can be assigned the type $a \to a$, or $b \to b$, or any other type of the same shape $A \to A$. But we can prove that all types assigned to a term t in the system $\mathtt{S}$ are equivalent by renaming, and are thus considered as one type — this standard result is called *uniqueness of typing*, and is proved by induction on structure of the term.

Proposition 1.4. *For any typing environment Γ and λ-term t, there is at most one type A such that $\Gamma \vdash t : A$ in* $\mathtt{S}$*, up to renaming of base types in A.*

The syntactic presentation used in this system allows to make the observation of Howard obvious: if λ-terms are erased from a typing rule, what we obtain is a valid inference rule of NJ. Therefore, there is a bijective correspondence between proofs of NJ and closed typing derivations. The last observation missing to establish the complete correspondence on the static level is that derivations are isomorphic to terms. This is a consequence of the structure of natural deduction proofs, where rule instances cannot be permuted.

Proposition 1.5. *For any λ-term t of type A, there is exactly one derivation of $\vdash t : A$.*

This $\mathtt{S}$ system can be extended in several ways. For example, the introduction of a conjunction in the logic allows to type terms where one can construct pairs of terms (u, v), and decompose pairs with the projection operators `fst` and `snd`. The typing rules for this extension are:

$$\mathtt{tup}\ \frac{\Gamma \vdash u : A \quad \Gamma \vdash v : B}{\Gamma \vdash (u, v) : A \wedge B} \qquad \mathtt{fst}\ \frac{\Gamma \vdash u : A \wedge B}{\Gamma \vdash \mathtt{fst}\ u : A} \qquad \mathtt{snd}\ \frac{\Gamma \vdash v : A \wedge B}{\Gamma \vdash \mathtt{snd}\ v : B}$$

Finally, beyond simple types, the introduction of quantifiers in the logic allows to handle polymorphism [GLT89], which makes explicit the fact that a function can be applied on different kind of arguments, provided they have a common shape that this function exploits. Quantification at second-order in NJ corresponds to the powerful System F, introduced by Girard and Reynolds [Gir71, Rey74].

[3] If pure λ-terms are considered, then several types can be assigned, and this is called the *Curry-style* presentation of the simply typed λ-calculus, as opposed to the *Church-style* presentation where type annotations are introduced in the terms, so that $\lambda x^A.x$ has type $A \to A$ while $\lambda x^B.x$ has type $B \to B$.

1.2 Detour Elimination as β-reduction

Now that we have described the syntactic correspondence between proofs in the NJ natural deduction and simply typed λ-terms, we can show how this correspondence extends to the dynamics of these objects. On the logical side, the dynamics of proofs is formed by the procedure of detour elimination, and a detour can be considered through the correspondence between proofs and typing derivations as follows:

$$\dfrac{\dfrac{\mathscr{P}_1}{\Gamma \vdash A} \quad \rightarrow_{\mathsf{i}} \dfrac{\overset{\mathscr{P}_2}{\Gamma, A \vdash B}}{\Gamma \vdash A \to B}}{\Gamma \vdash B} \rightarrow_{\mathsf{e}} \quad \equiv \quad \dfrac{\dfrac{u}{\Gamma \vdash u : A} \quad \mathtt{lam}\, \dfrac{\overset{t}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x.t : A \to B}}{\Gamma \vdash (\lambda x.t)\, u : B}\, \mathtt{app}$$

The elimination of detours in a proof in NJ, as described in Chapter 1, consists in the replacement of all axioms in $\mathscr{P}_2$ involving the A with the proof $\mathscr{P}_1$ of A. This can in turn be observed as a transformation of typing derivations within the S type system, and by equivalence of closed derivations and λ-terms, as a transformation of λ-terms. Since the variables correspond to instances of the `var` typing rule, and therefore to axiom instances, this replacement can be read as the replacement of all occurrences of x by the term u, as illustrated below:

$$\dfrac{\dfrac{u}{\Gamma \vdash u : A} \quad \mathtt{lam}\, \dfrac{\overset{t}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x.t : A \to B}}{\Gamma \vdash (\lambda x.t)\, u : B}\, \mathtt{app} \quad \longrightarrow \quad \overset{t\{u/x\}}{\Gamma \vdash t\{u/x\} : B}$$

The elimination of one detour in a proof in NJ thus corresponds to one single β-reduction step in a λ-term, where a function $\lambda x.t$ is applied to its argument u by replacing all occurrences of the formal argument x in the body t of the function by the term u, which is written $(\lambda x.t)\, u \longrightarrow_\beta t\{u/x\}$. However, the result stating that all the detours can be eliminated from a given proof in NJ does not necessarily ensures that any reduction strategy will be able to remove all redexes from a term, but it provides a weaker result, the existence of such a strategy.

Theorem 1.6. *Given a simply typed λ-term t, there exists a terminating strategy that allows to compute the strong β-normal form of t.*

Proof. By detour elimination in NJ. Indeed, since it was proved that all the detours can be eliminated from any given proof in NJ, through the correspondence shown above all β-redexes can be eliminated from the simply typed λ-term t. The strategy depends on the proof of detour elimination: from the proof given in Chapter 1, the strategy extracted consists in reducing first the β-redexes $(\lambda x.u)\, v$ such that u and v are already in normal form, then reducing newly created redexes, and so on. □

The more general result that we would like to obtain is the guarantee that the complete reduction of a typeable λ-term always produces its normal form, rather than looping forever if some *» wrong choice «* of redex was made at some point in the process. This is called *strong normalisation*, by opposition to the *weak normalisation* result that we have proved through the correspondence between detour elimination and β-reduction, and this amounts to the termination of a subset of the calculus, obtained by restricting terms to the simply typed ones.

Proposition 1.7. *Strong β-reduction terminates on simply typed λ-terms.*

Unfortunately, this theorem is quite complicated to prove. There is no direct, simple proof, and the usual way of presenting this is to observe that it is a particular case of the strong normalisation result for System F — the polymorphic λ-calculus, a powerful extension of the standard λ-calculus —, which was proved by semantic means, using a the technique of *» reducibility candidates «*, due to Girard. There are other proofs, such as the one by Gandy [Gan80], but they rely on a conceptually complex interpretation of typeis rather than the assignement of a measure to typed terms, that would decrease at each β-reduction step. From the logical viewpoint, a generalisation of the weak normalisation result shown above would require the definition of a detour elimination procedure in which any detour can be picked and reduced, at any step, but the measure used to control the induction would probably be quite complex.

In terms of functional programming, the normalisation result is at the heart of the guarantees offered by the *proofs-as-programs* methodology. Indeed, the point of introducing type systems built based on logical systems into real compilers is that *» well-typed terms never go wrong «*, which means in a purely applicative language that no program can pass type-checking and enter an infinite loop when executed.

The extensions made to the basic S type system follow the same scheme as the basic detour elimination correspondence. For example, the use of the pair syntax, as shown previously, creates a new situation similar to a usual detour, but involving a pair of conjunction introduction and elimination instances. The reduction rules on pairs:

$$\begin{array}{ll} \mathtt{fst}\,(u,v) & \longrightarrow_{\pi_1} \quad u \\ \mathtt{snd}\,(u,v) & \longrightarrow_{\pi_2} \quad v \end{array}$$

correspond in this situation to the reduction of a conjunctive detour, when one of the elimination rules is used below an introduction and allows to choose one of the branches, to be kept.

$$\mathtt{app}\,\dfrac{\mathtt{app}\,\dfrac{\overset{u}{\Gamma \vdash u : A} \quad \overset{v}{\Gamma \vdash v : B}}{\Gamma \vdash (u,v) : A \wedge B}}{\Gamma \vdash \mathtt{fst}\,(u,v) : A} \quad \longrightarrow \quad \overset{u}{\Gamma \vdash u : A}$$

The normalisation result in this setting ensures that any typed term will reduce properly, and not be blocked in some invalid situation, for example where the `fst` operator would be applied on a function rather than a pair.

2 Cut Elimination and Explicit Substitutions

As mentioned in Chapter 1, the substitutive detour elimination procedure is not so informative, leaving the whole substitution process unspecified. This situation is improved by the use of the permutative procedure, where substitution is explicitly described in terms of cut instances permuted upwards until they meet a matching instance. As suggested by the vocabulary we used, this is equivalent, through the principles of the Curry-Howard correspondence, to the use of explicit substitutions as an implementation of β-reduction in the λ-calculus. Indeed, the cut rule is the embodiment of a » *detour* « and its use can be formally identified to the use of an explicit substitution, both in standard λ-calculi and in pure explicit substitutions calculi — that is, in natural deduction and in the sequent calculus, respectively. In terms of typing, the cut rule is used as the following rule:

$$\mathsf{cut}\ \frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \qquad \equiv \qquad \mathtt{sub}\ \frac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash t[x \leftarrow u] : B}$$

and all variants of the cut rule correspond to different ways of handling an explicit substitution, for example regarding its duplication, in the mutlitplicative or in the additive presentation.

2.1 The λ-calculus with Explicit Substitutions

The standard variants of the λ-calculus using explicit substitutions are conservative extensions that refine the standard λ-calculus, in the sense that they allow to deal with usual terms but introduce a new syntax for terms that are intermediate steps in the computation. Thus, we start as usual with a countable set of variables, denoted by small latin letters such as x, y and z.

Definition 2.1. *The language of* standard λ-calculi with explicit substitutions *is an extension of the language of the λ-calculus, defined by the following grammar:*

$$t, u ::= x \mid \lambda x.t \mid t\ u \mid t[x \leftarrow u]$$

where the object $[x \leftarrow u]$ is a binder for the variable x in a term t.

Such λ-calculi of explicit substitutions have been introduced [ACCL90] in the setting of the *name-free* λ-calculus, using *deBruijn indices* [dB72] instead of names, because of their relation to the implementation of functional languages or theorem provers. However, they have been adapted to use names, which are more readable for humans than indices — many variants can be found in the literature[4]. For the sake of simplicity, we will only consider calculi with names.

The notion of *binding* is central in the λ-calculus and all of its extensions. Here, it is extended so that explicit substitutions are also binders for variables, in addition to abstractions. This should be understood as the case where the *actual argument* of the function has been attached to the name that was the *formal argument*, while

[4] An overview of the jungle of λ-calculi with explicit substitutions can be found in [Kes07].

the actual argument is still unknown in the case of an abstraction not yet coupled with an application. The explicit substitution » *object* « is meant to internalise the standard notion of substitution, which is defined as usual in the λ-calculus.

Definition 2.2. *Given terms t and u, the* implicit substitution *of u for x in t, denoted by $t\{u/x\}$, is t where all occurrences of x have been simultaneously replaced with u.*

The replacement needs to be simultaneous because the variable x could appear inside u itself, so that incremental replacement could enter a loop. Notice in general that this definition is problematic when u contains variable names that are also used in t but do not refer to the same object, because they are introduced by different binders, since it would result in a term where these initially distinct variables are identified — this is called a *capture* of variable. It is possible to refine the definition of substitution into one of *capture-avoiding* substitution, but it is also sufficient to ensure that different names will be used for different variables.

Definition 2.3. *Given a term t, the set of its* free variables*, denoted by* $\mathrm{fv}(t)$*, and the set of its* bound variables*, denoted by* $\mathrm{bv}(t)$*, are defined as follows:*

$$\begin{aligned}
\mathrm{fv}(x) &= \{x\} & \mathrm{bv}(x) &= \emptyset \\
\mathrm{fv}(\lambda x.t) &= \mathrm{fv}(t) \setminus \{x\} & \mathrm{bv}(\lambda x.t) &= \mathrm{bv}(t) \cup \{x\} \\
\mathrm{fv}(t\ u) &= \mathrm{fv}(t) \cup \mathrm{fv}(u) & \mathrm{bv}(t\ u) &= \mathrm{bv}(t) \cup \mathrm{bv}(u) \\
\mathrm{fv}(t[x \leftarrow u]) &= (\mathrm{fv}(t) \setminus \{x\}) \cup \mathrm{fv}(u) & \mathrm{bv}(t[x \leftarrow u]) &= \mathrm{bv}(t) \cup \{x\} \cup \mathrm{bv}(u)
\end{aligned}$$

Moreover, we write $x \in t$ to denote the fact that a variable x appears in some term t. A complete functional program is not supposed to use unknown variables, and it is often required to work only with *closed terms*, which are terms t such that $\mathrm{fv}(t) = \emptyset$, or equivalently such that if $x \in t$, then $x \in \mathrm{bv}(t)$.

Now, we can define an equational theory on terms which allows to solve the problem of clashes among variable names by renaming them when needed. Given a term, we will build an equivalent term where some variables have been renamed using fresh names and thus cannot be captured during substitution.

Definition 2.4. *The congruence denoted by $\equiv_\alpha$ and called α-conversion is the smallest reflexive, symmetric and transitive relation such that:*

$$\begin{aligned}
\lambda x.t &\equiv_\alpha \lambda y.u && \text{if } t \equiv_\alpha v\{x/y\}, \text{ where } y \notin \mathrm{bv}(v) \text{ and } u \equiv_\alpha v \\
t\ u &\equiv_\alpha s\ v && \text{if } t \equiv_\alpha s \text{ and } u \equiv_\alpha v \\
t[x \leftarrow u] &\equiv_\alpha s[y \leftarrow v] && \text{if } t \equiv_\alpha r\{x/y\} \text{ and } u \equiv_\alpha v, \text{ where } y \notin \mathrm{bv}(r) \text{ and } s \equiv_\alpha r
\end{aligned}$$

In the following, we will always consider terms *modulo* α-conversion so that any variable is bound at most once in any given term, and new names introduced are always fresh — an assumption often called *Barendregt's convention* [Bar84]. This simplifies notations by avoiding some side conditions that are thus made implicit whenever the situation is not ambiguous — for example, if the names x and y are used in a term, they should be considered distinct. Another consequence of this convention is that free variables and bound variables are always disjoint sets, and we can refine the statement $x \in t$ by considering $|t|_x$, the number of occurrences of some variable x in a term t.

Remark 2.5. *The treatement of terms modulo renaming is necessary only because we use names to represent variables, rather than other means such as* de Bruijn indices*, where variables are identified through the structure of the term itself [dB72].*

In order to provide an operational semantics for this language, a set of reduction rules must be defined, and we expect this reduction system to be able to simulate somehow the β-reduction rule, since the goal is in the end to refine the evaluation process of functional programs, seen as λ-terms. There are many different ways of achieving this, although the basic ideas are always the same, the difference being a matter of refinement. We will describe here some of the standard solutions, from the most basic ones to more elaborate solutions where erasure and duplication of substitutions are handled with care. All the λ-calculi presented in this section have been described in the literature, often several times under a form or another, with names or without, and are described in different surveys [Les94, Kes07].

We are interested in *strong reduction*, in the sense that reduction rules can be applied anywhere in a term, and in particular under abstractions — that is, within the body of a function. It is more complex than *weak reduction*, but is necessary in many cases, for example when building proof assistants, where equality requires to fully reduce the two terms to be compared. In all the reduction systems presented below, we assume that the following basic set of rules is implicitly used:

$$\begin{array}{rcll} \lambda x.t & \longrightarrow & \lambda x.v & \text{if } \ t \longrightarrow v \\ t\ u & \longrightarrow & v\ u & \text{if } \ t \longrightarrow v \\ t\ u & \longrightarrow & t\ v & \text{if } \ u \longrightarrow v \\ t[x \leftarrow u] & \longrightarrow & v[x \leftarrow u] & \text{if } \ t \longrightarrow v \\ t[x \leftarrow u] & \longrightarrow & t[x \leftarrow v] & \text{if } \ u \longrightarrow v \end{array} \tag{3}$$

which is equivalent to say that reduction rules can be applied anywhere inside the term. The treatment of weak reduction, which is used for evaluation in functional programming languages — where the body of a function is not reduced in advance, except for some optimisations —, only requires to remove the first rule.

Basic implementation. The most basic system of explicit substitutions is the naïve approach where substitutions are pushed inside the term and duplicated each time there are two subterms, until they meet a variable. In this situation, there are two possible cases: either this variable is the one bound by the substitution, and a direct replacement is performed, or it is not and the substitution is erased. This system is known as the $\lambda\mathtt{x}$-calculus [BR95], for which the operational behaviour is defined by the reduction rules given in Figure 2 — actually, the contextual closure of these rules, or equivalently these rules extended with the basic system that was shown in (3). This reduction system will denoted by $\longrightarrow_{\lambda\mathtt{x}}$ from now on.

This calculus is only using explicit substitutions in a weak way, in the sense that such substitutions cannot be composed. This implies that substitutions cannot be carried out independently, so that the reduction mechanism defined there closely resembles plain β-reduction, extended with the complete definition of what the substitution operation means — that is, how to replace all occurrences of a variable with a term. This restriction is the key to prove in a simple way that $\lambda\mathtt{x}$ has good properties with respect to the λ-calculus.

$$
\begin{array}{rcl}
(\lambda x.t)\,u & \longrightarrow_{\mathtt{B}} & t[x \leftarrow u] \\
x[x \leftarrow u] & \longrightarrow_{\mathtt{var}} & u \\
y[x \leftarrow u] & \longrightarrow_{\mathtt{nov}} & y \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{app}} & t[x \leftarrow u]\ v[x \leftarrow u]
\end{array}
$$

Figure 2: Reduction rules of the $\lambda\mathtt{x}$-calculus

Relation to λ. A basic, important property is the ability of a calculus to simulate the β-reduction rule of the λ-calculus. Indeed, calculi with explicit substitutions have been introduced to refine the operational behaviour of the λ-calculus, and in the end the goal is to implement functional programming languages, which are described informally in terms of functions evaluated by β-reduction. This requires to have a translation $[\![\cdot]\!]^{\lambda}_{\mathtt{x}}$ from λ-terms to terms with explicit substitutions, but in the case of the standard explicit syntax, as used in $\lambda\mathtt{x}$, this translation is actually the identity — the syntax here is a direct extension of the usual λ-terms syntax.

The standard technique to prove this simulation result is to show that the notion of explicit substitution defined by the reduction rules corresponds exactly to the usual *meta-substitution* operation — the so-called *full composition* result. However, the lack of composition for substitutions in $\lambda\mathtt{x}$ disallows to prove this in general, but it can be proved for substitutions applied on *pure terms*, those $\lambda\mathtt{x}$-terms that happen to be λ-terms, because they use no explicit substitution.

Proposition 2.6. *For any given λ-terms t and u, we have $t[x \leftarrow u] \longrightarrow^{*}_{\lambda\mathtt{x}} t\{u/x\}$.*

Of course, this can work in such a simple way only because any λ-term is also a valid term in $\lambda\mathtt{x}$, and this implies that the simulation here is not as strong a result as it is in a system with composition. The idea of simulation is to turn a β-redex into an explicit substitution and then carry out completely this substitution, before treating any other β-redex.

Proposition 2.7 (Simulation in $\lambda\mathtt{x}$)**.** *Given t and u, if $t \longrightarrow_{\beta} u$ then $t \longrightarrow^{*}_{\lambda\mathtt{x}} u$.*

The other way around, we need to make sure that the result of this simulation is meaningful, by proving that reductions performed in the $\lambda\mathtt{x}$-calculus always reflect parts of reductions in the λ-calculus. For this, we need another translation.

Definition 2.8. *The translation $[\![\cdot]\!]^{\mathtt{x}}_{\lambda}$ from $\lambda\mathtt{x}$-terms into λ-terms is defined as:*

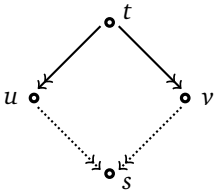
$$
\begin{array}{rclcrcl}
[\![x]\!]^{\mathtt{x}}_{\lambda} & = & x & \qquad & [\![t\,u]\!]^{\mathtt{x}}_{\lambda} & = & [\![t]\!]^{\mathtt{x}}_{\lambda}\,[\![u]\!]^{\mathtt{x}}_{\lambda} \\
[\![\lambda x.t]\!]^{\mathtt{x}}_{\lambda} & = & \lambda x.[\![t]\!]^{\mathtt{x}}_{\lambda} & & [\![t[x \leftarrow u]]\!]^{\mathtt{x}}_{\lambda} & = & [\![t]\!]^{\mathtt{x}}_{\lambda}\{[\![u]\!]^{\mathtt{x}}_{\lambda}/x\}
\end{array}
$$

The idea is that we can establish a correspondence, in both directions, between reductions in the standard λ-calculus and in our calculus with explicit substitution.

Proposition 2.9 (Projection of $\lambda\mathtt{x}$)**.** *For t, u, if $t \longrightarrow_{\lambda\mathtt{x}} u$ then $[\![t]\!]^{\mathtt{x}}_{\lambda} \longrightarrow^{*}_{\beta} [\![u]\!]^{\mathtt{x}}_{\lambda}$.*

These two results of simulation and projection, together, ensure that we defined a reasonable set of reduction rules, at least with respect to its goal of implementing the λ-calculus with a finer-grained operational behaviour.

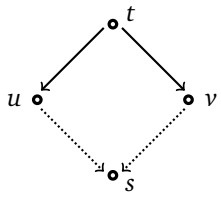
Confluence. Another important property one can expect from a calculus where the operational behaviour is expressed by reduction rules is *confluence* — or even its generalisation, called the *Church-Rosser property* [CR36] —, which states that if a term t reduces to two different terms u and v, then these terms can themselves be reduced to a unique term s, as illustrated by the following diagram:



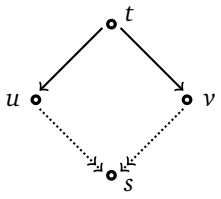
(confluence)

In such a diagram, we denote terms with points and reductions with arrows, where a single headed arrow indicates one-step reduction while a double headed arrow indicates several steps of reductions. A solid arrow represents an assumption while a dotted one represents the conclusion of a diagramatic statement. Also, links with no arrow end denotes a reduction with at most one step, but possibly none.

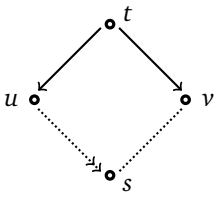
Showing that a given calculus is confluent is not always immediate, and it may prove useful to use other similar properties to finally obtain this result[5]. The most common ones are illustrated in the following diagrams:



(diamond property)



(local confluence)



(strong confluence)

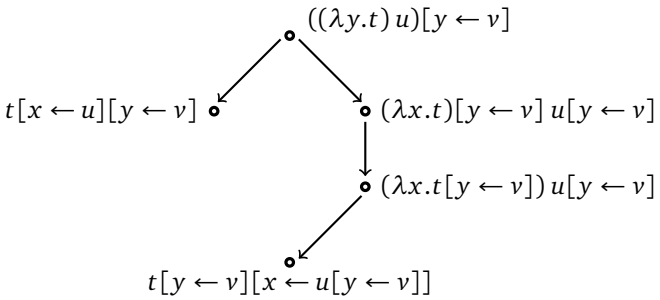
These properties are not equivalent[6]: the diamond property here is clearly the strongest one, and it trivially implies confluence, but strong confluence also implies confluence[7]. Finally, the local confluence does not imply the standard confluence in the general case, but does if the reduction system is terminating, as ensured by Newman's Lemma [New42]. An » *interesting* « calculus usually does not enjoy the diamond property, since reduction rules often tend to interfere with each other. But a standard technique for proving confluence, called *parallel reductions* [Bar84], is to design another system where several independent steps can be performed at the same time, and prove that this reduction system has the diamond property. Then, if we can show that the two systems have the same transitive closure, we can deduce that the original system is confluent.

[5]The standard proofs of confluence that can be found in the literature usually take advantage of the connection between confluence and other similar properties.

[6]They are explained in more details in [Sel07], in the setting of the pure λ-calculus.

[7]Although one should be careful there and handle both of the two symmetric cases [BN98].

In the case of the $\lambda\mathtt{x}$-calculus, the reduction system is simple, but the diamond property does not hold, because of only one situation where the diagram cannot be closed — this case shows that strong confluence does not hold either:



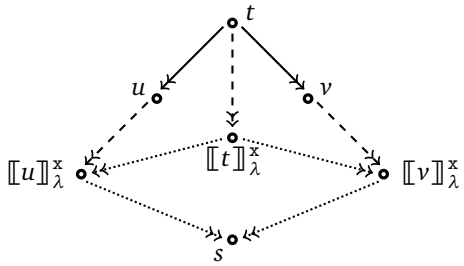
But this is not a problem that can be solved by using the parallel reductions technique, since it is induced by the lack of rules for composition of substitutions. We will thus rely on the fact that the standard λ-calculus is confluent, and use the correspondence we just established to prove that $\lambda\mathtt{x}$ is also confluent. For this, we need a lemma stating that the translation of a term can be obtained by reduction.

Proposition 2.10. *For any term t, we have $t \longrightarrow^{*}_{\lambda\mathtt{x}} [\![t]\!]^{\mathtt{x}}_{\lambda}$.*

The ability to reduce a term to its translation is crucial, since it provides a way to connect the confluence result of the λ-calculus to the situation of $\lambda\mathtt{x}$. This means that a diagram can be closed by first reducing terms to their translations, and then reusing the closing reductions from the standard λ-calculus.

Proposition 2.11. *The $\lambda\mathtt{x}$-calculus is confluent.*

The idea of the proof of confluence is expressed by the diagram below, where one can observe that it relies on the correspondence between $\lambda\mathtt{x}$ and the standard λ-calculus to lift the result from the standard setting into the refined setting of $\lambda\mathtt{x}$, using explicit substitutions:



PSN. In the process of designing a calculus of explicit substitutions refining the standard λ-calculus, there is an important property to ensure, which concerns the translation of reduction sequences. This is the *preservation of strong normalisation*, often called PSN, which states that if a given λ-term t is strongly normalising, then its translation, $[\![t]\!]^{\lambda}_{\mathtt{x}}$ in the case of $\lambda\mathtt{x}$, which is exactly t, should also be a strongly normalising term, in the target calculus.

In the λ-calculus, and in most of its variants, such as $\lambda\mathtt{x}$, the *normal form* of a term t — that is, a term u such that $t \longrightarrow^* u$ and u cannot be reduced — is unique whenever it exists, because of confluence. What we call a strongly normalising term is then a t such that all possible reduction sequences starting from t are finite, and thus end with its normal form, as described in the following definition[8].

Definition 2.12. *The set* $\mathrm{SN}_{\lambda\mathtt{x}}$ *of* strongly normalising $\lambda\mathtt{x}$*-terms is the smallest set such that* $t \in \mathrm{SN}_{\lambda\mathtt{x}}$ *if* t *is in normal form or* $t \longrightarrow_{\lambda\mathtt{x}} u$ *with* $u \in \mathrm{SN}_{\lambda\mathtt{x}}$.

The PSN property is rather complex to prove, although it is significantly simpler in $\lambda\mathtt{x}$ than in more refined calculi. Several methods have been defined to establish this result, relying on the idea that pushing an explicit substitution inside another term should always participate to the process of implementing β-reduction through small, decomposed steps and never add any additional computation, that would be completely unrelated to this task — and would thus potentially introduce loops in the implementation of a strongly normalising λ-term.

Proposition 2.13 (PSN in $\lambda\mathtt{x}$)**.** *Given a* λ*-term* t*, if* $t \in \mathrm{SN}_\beta$ *then* $t \in \mathrm{SN}_{\lambda\mathtt{x}}$.

This result confirms that we can use the $\lambda\mathtt{x}$-calculus as an implementation of the standard λ-calculus, but the question remains now to know if this is an efficient implementation, or if we should try to design a more subtle implementation of the substitution operation. Unfortunately, reduction in this calculus induces the copy of whole subterms, of arbitrary size, each time a substitution crosses an application, which is very inefficient. Moreover, it disallows any kind of interaction between two explicit substitutions, although we would like to either exchange them, or compose them, as follows:

$$\begin{array}{rcl} t[x \leftarrow u][y \leftarrow v] & \longrightarrow & t[y \leftarrow v][x \leftarrow u] \\ t[x \leftarrow u][y \leftarrow v] & \longrightarrow & t[x \leftarrow u[y \leftarrow v]] \end{array}$$

However, these operations are dangerous to introduce as reduction rules in the system. Indeed, the first one is a valid exchange of explicit substitutions only if y does not appear as a free variable in u, since without this condition this rule would break the scoping structure of the term. The second operation is also problematic, since it can lead to non-termination of terms that are actually the translation of a strongly normalising λ-term, as illustrated by the famous counter-example given by Melliès [Mel95]. It can be observed that proving the PSN property in the presence of composition is more complicated than in $\lambda\mathtt{x}$, since if we add this reduction rule, some subterms can interact in a way that does not correspond to the β-reduction process, and induce non-termination.

Example 2.14. *Consider the term* $t = (\lambda z.z)[x \leftarrow y\ y][y \leftarrow \delta]$*, where* δ *is a term such that* $\Omega = \delta\ \delta$ *is not terminating. Then, we have* $[\![t]\!]^{\mathtt{x}}_{\lambda} = \lambda z.z$*, all the subterms of* t *have strongly normalising translations, and thus* t *is strongly normalising in* $\lambda\mathtt{x}$*, but if we add the rule for composition described above, we have the new reduction sequence* $t \longrightarrow^* (\lambda z.z)[x \leftarrow \Omega]$*, which yields a term that is not strongly normalising.*

[8] The set of strongly normalising terms in any calculus defined through a reduction $\longrightarrow_\bullet$ is defined following the same scheme as the one used for $\lambda\mathtt{x}$, and it is denoted by $\mathrm{SN}_\bullet$ as usual.

$$
\begin{array}{rcll}
(\lambda x.t)\, u & \longrightarrow_{\mathtt{B}} & t[x \leftarrow u] & \\
x[x \leftarrow u] & \longrightarrow_{\mathtt{var}} & u & \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{not}} & t & (x \notin t) \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] & (x \in t) \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apl}} & t[x \leftarrow u]\ v & (x \in t, x \notin v) \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apr}} & t\ v[x \leftarrow u] & (x \notin t, x \in v) \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apb}} & t[x \leftarrow u]\ v[x \leftarrow u] & (x \in t, x \in v) \\
t[x \leftarrow u][y \leftarrow v] & \longrightarrow_{\mathtt{cmp}} & t[x \leftarrow u[y \leftarrow v]] & (y \notin t, y \in u) \\
t[x \leftarrow u][y \leftarrow v] & \longrightarrow_{\mathtt{cpp}} & t[y \leftarrow v][x \leftarrow u[y \leftarrow v]] & (y \in t, y \in u)
\end{array}
$$

$$
t[x \leftarrow u][y \leftarrow v] \equiv_e t[y \leftarrow v][x \leftarrow u] \qquad (x \notin v, y \notin u)
$$

Figure 3: Reduction rules and equation of the $\lambda\mathtt{es}$-calculus

Refined calculi. A solution to the problem of introducing the additional rules for the composition of substitutions without losing good properties of the calculus, is to handle with care erasure, duplication and distribution of substitutions. This is what is done for example in the $\lambda\mathtt{es}$-calculus [Kes07], which uses the presence of variables in a subterm as a condition for the application of some reduction rules. Such a design of the rules allows to avoid useless duplications of substitutions.

The syntax for the $\lambda\mathtt{es}$-calculus is again the standard syntax of λ-calculi with explicit substitutions, and its reduction rules are given in Figure 3 — we use again the contextual closure of the rules, or the basic system shown in (3). This reduction system will be denoted by $\longrightarrow_{\lambda\mathtt{es}}$ from now on.

This calculus is a refinement of the $\lambda\mathtt{x}$-calculus, which uses the ability to look at the set of free variables of a given term to decide whether a substitution should be pushed inside another construction or not. One consequence of this ability is that the *garbage collection* operation, that the `nov` rule embodies in the $\lambda\mathtt{x}$-calculus, can be generalised into the `not` rule, which erases the substitution before it is pushed further inside the term, if it binds an unused variable. The rule dealing with some substitution applied on an application is also refined into three rules, that shoud be used depending on the use of the bound variable in the subterms.

Finally, the $\lambda\mathtt{es}$-calculus is equipped with two composition rules, that are also used depending on the use of the variable bound by the substitution, and also one equation which embodies the case where two unrelated substitutions are applied on the same term. This rewriting is expressed as an equation, and not implemented as another reduction rule, because it would otherwise introduce an obvious looping mechanism, as mentioned before. The reduction system is extended into $\longrightarrow^{\equiv}_{\lambda\mathtt{es}}$ by considering reductions $t \longrightarrow^{\equiv}_{\lambda\mathtt{es}} u$ of the shape $t \equiv v \longrightarrow_{\lambda\mathtt{es}} w \equiv u$, where $\equiv$ is the smallest congruence induced by alpha-equivalence and the $\equiv_e$ relation. Composition allows to prove the full composition result.

Proposition 2.15 (Full composition in $\lambda\mathtt{es}$)**.** *For t and u, $t[x \leftarrow u] \longrightarrow^{\equiv *}_{\lambda\mathtt{es}} t\{u/x\}$.*

Simulation, confluence, PSN. This result is stronger than its equivalent in $\lambda\mathtt{x}$, as substitutions are carried out independently, by crossing substitutions or moving inside the body of another substitution when needed. However, simulation in the setting of $\lambda\mathtt{es}$ is not using the full power of the reduction system, although the complete β-reduction of different redexes could be implemented in many different ways. Projection is also handled the same way as in the $\lambda\mathtt{x}$-calculus, through a translation $[\![\cdot]\!]^{\mathtt{es}}_{\lambda}$ which is defined the same way as $[\![\cdot]\!]^{\mathtt{x}}_{\lambda}$.

Since the basic results of simulation and projection relating reduction in $\lambda\mathtt{es}$ to β-reduction hold, we can use the same technique as for the $\lambda\mathtt{x}$-calculus to prove that the $\lambda\mathtt{es}$-calculus is confluent, through the lemma showing that translation can be obtained by reduction. Regarding all of these results, $\lambda\mathtt{es}$ is therefore not much different from $\lambda\mathtt{x}$.

As mentioned, the PSN property is more difficult to prove in the context of a calculus allowing the composition of explicit substitutions. This result is usually established through translations into other calculi, such as those equipped with an explicit erasure operator [Kes07], and the λI-calculus [Klo80] where any weakly normalising term is also a strongly normalising one. An important observation in the study of normalisation in the $\lambda\mathtt{es}$-calculus is the distinction between the B rule and the » *substitution pushing* « subsystem $\longrightarrow^{\equiv}_{\mathtt{es}}$, which can be shown strongly normalising through the definition of an appropriate measure.

Proposition 2.16. *The reduction system* $\longrightarrow^{\equiv}_{\mathtt{es}}$ *is strongly normalising.*

Notice that this is not sufficient to obtain any normalisation result on the whole reduction system $\longrightarrow^{\equiv}_{\lambda\mathtt{es}}$, since the B rule is crucial in its role of manipulating the distinction in a term between the β-redexes of this term and explicit substitutions.

Proposition 2.17 (PSN in $\lambda\mathtt{es}$)**.** *Given a λ-term t, if $t \in \mathrm{SN}_{\beta}$ then $t \in \mathrm{SN}_{\lambda\mathtt{es}}$.*

This result is quite important, since it means that we can design such a complex implementation of the λ-calculus while retaining the most important properties of programs written in this language — although the mechanism of composition of substitutions can induce highly complex behaviours in the reduction of terms.

Other calculi. There are several different possibilities in the design of λ-calculi with explicit substitutions, in particular depending on the choices made to handle erasure and duplication of the substitutions. For instance, a refinement of $\lambda\mathtt{es}$ can be obtained by separating the duplication operation from the structure of the term under the substituition. Such a calculus, called the $\lambda\mathtt{s}$-calculus [Ren11], can be described by the rules given in Figure 4, where $t_{[y/x]}$ denotes the term t where exactly one occurrence of x has been replaced with y. The operation of duplicating a substitution is here implemented only in the special dup rule, but this calculus behaves like $\lambda\mathtt{es}$ and it can be proved that it has all the same good properties of simulation, confluence and preservation of strong normalisation [KR11].

A further step in the refinement of such λ-calculi with explicit substitutions is the introduction of *resource operators*, in order to handle explicitly the erasure and duplication of explicit substitutions. This was done for erasure only in the calculus with *garbage collection* $\lambda\mathtt{xgc}$ [BR95], and it was later generalised to both erasure and duplication in the $\lambda\mathtt{lxr}$-calculus [KL05].

$$
\begin{array}{rcll}
(\lambda x.t)\,u & \longrightarrow_{\mathtt{B}} & t[x \leftarrow u] & \\
x[x \leftarrow u] & \longrightarrow_{\mathtt{var}} & u & \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{not}} & t & (|t|_x = 0) \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{dup}} & t_{[y/x]}[x \leftarrow u][y \leftarrow u] & (|t|_x \geq 2) \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] & (|t|_x \geq 1) \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apl}} & t[x \leftarrow u]\ v & (|t|_x \geq 1, |v|_x = 0) \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apr}} & t\ v[x \leftarrow u] & (|t|_x = 0, |v|_x \geq 1) \\
t[x \leftarrow u][y \leftarrow v] & \longrightarrow_{\mathtt{cmp}} & t[x \leftarrow u[y \leftarrow v]] & (|t|_y = 0, |u|_y \geq 1)
\end{array}
$$

$$
t[x \leftarrow u][y \leftarrow v] \equiv_e t[y \leftarrow v][x \leftarrow u] \qquad (x \notin v, y \notin u)
$$

Figure 4: Reduction rules and equation of the $\lambda\mathtt{s}$-calculus

In such a calculus, an explicit substitution on x can be erased or duplicated only when it meets a term where the related operator on x appears at toplevel. In the syntax of $\lambda\mathtt{lxr}$, the rules are:

$$
\begin{array}{rcl}
(\mathscr{W}_x.t)[x \leftarrow u] & \longrightarrow & \mathscr{W}_\phi.t \\
(\mathscr{C}_x^{y,z}.t)[x \leftarrow u] & \longrightarrow & \mathscr{W}_\phi^{\psi,\mu}.t[y \leftarrow u_\psi][z \leftarrow u_\mu]
\end{array}
$$

where ϕ is the list $\mathrm{fv}(u)$ and u_ψ and u_μ are copies of the term u where all the free variables have been replaced by fresh variables from the lists ψ and μ.

2.2 Cut in Natural Deduction and Explicit Substitutions

The cut rule is used in NJ as a tool to perform detour elimination, but it can also be given the normal status of rule. The permutative proof of detour elimination in NJ can then be trivially interpreted as a normalisation result in $\mathsf{NJ} \cup \{\mathsf{cut}\}$, where two different kinds of redexes are targeted, standard detours and cut instances. In this setting, the dynamics of proofs and programs are refined:

Logic	**Computation**
proof $\mathscr{P}$ of A in NJ	closed, pure $\lambda\mathtt{x}$-term t of type A
proof $\mathscr{P}$ of A in $\mathsf{NJ} \cup \{\mathsf{cut}\}$	closed $\lambda\mathtt{x}$-term t of type A
cut on B in $\mathscr{P}$	redex $u[x \leftarrow v]$ in t, with v of type B
detour elimination in $\mathscr{P}$	strong reduction $t \longrightarrow^*_{\lambda\mathtt{x}} r$
elimination of a cut in $\mathscr{P}$	reduction $u[x \leftarrow v] \longrightarrow^*_{\mathtt{x}} u\{v/x\}$ in t

where the computational device used in the $\lambda\mathtt{x}$-calculus, which corresponds to the additive presentation of NJ. The refinement of the pure λ-calculus into $\lambda\mathtt{x}$ is there an equivalent to the extension of NJ with the cut rule.

On a syntactic level, the S type system is extended into the Sx system by adding the sub rule shown above. This rule expresses the idea that in order to assign type B to the term $t[x \leftarrow u]$, one must be able to assign some type A to u and type t with B under the assumption that the variable x has type A inside t. This extension does not change the way other syntactic constructs are handled, and all the notions used in S are used the same way in Sx. This system is also syntax-directed, as the explicit substitution $[x \leftarrow u]$ is given the status of valid construct in the syntax of terms, and Sx has the same properties as S concerning the static level of typing derivations. In particular:

- The typing process in Sx is terminating.
- A λx-term t has at most one type in an environment Γ in Sx, up to renaming.
- There is exactly one derivation of $\vdash t : A$ in Sx, for any λx-term t of type A.

These results can be proved the same way as they are proved in S, with some minor adaptations to accomodate the cut. Typing is essentialy performed the same way in S and in the Sx system. The dynamics of proofs seen as programs, however, is described differently in this setting. The correspondence between local rewritings of proof objects, by permutation of cuts, and the reduction steps used in λx can be observed by interpreting each step[9] independently:

1. The reduction rule $(\lambda x.t)\, u \longrightarrow_{\mathrm{B}} t[x \leftarrow u]$ corresponds to the transformation of a detour into a cut instance:

$$\mathtt{app}\,\dfrac{\Gamma \vdash u : A \qquad \mathtt{lam}\,\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}}{\Gamma \vdash (\lambda x.t)\, u : B} \quad \longrightarrow \quad \mathtt{sub}\,\dfrac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash t[x \leftarrow u]}$$

2. The reduction rule $x[x \leftarrow u] \longrightarrow_{\mathtt{var}} u$ corresponds to the replacement of an axiom by the proof of the lemma involved in the cut:

$$\mathtt{sub}\,\dfrac{\Gamma \vdash u : A \qquad \mathtt{var}\,\dfrac{}{\Gamma, x : A \vdash x : A}}{\Gamma \vdash x[x \leftarrow u] : A} \quad \longrightarrow \quad \Gamma \vdash u : A$$

3. The reduction rule $y[x \leftarrow u] \longrightarrow_{\mathtt{nov}} y$ corresponds to the erasure of the cut and of the proof of the lemma, when an axiom on another formula than the lemma A is encountered:

$$\mathtt{sub}\,\dfrac{\Gamma, y : B \vdash u : A \qquad \mathtt{var}\,\dfrac{}{\Gamma, y : B, x : A \vdash y : B}}{\Gamma, y : B \vdash y[x \leftarrow u] : B} \quad \longrightarrow \quad \mathtt{var}\,\dfrac{}{\Gamma, y : B \vdash y : B}$$

[9] We show here only the part of the typing derivation being rewritten, since all the transformations involved are local, and proofs of the premises can be reused after transformation — and we show no case involving con, since it is similar to var, and it is not necessary to have constants.

4. The reduction rule $(\lambda y.t)[x \leftarrow u] \longrightarrow_{\mathtt{lam}} \lambda y.t[x \leftarrow u]$ corresponds to the permutation of the cut above an introduction instance, so that the derivation:

$$\mathtt{sub}\,\frac{\Gamma \vdash u : A \qquad \mathtt{lam}\,\dfrac{\Gamma, x : A, y : C \vdash t : D}{\Gamma, x : A \vdash \lambda y.t : C \to D}}{\Gamma \vdash (\lambda y.t)[x \leftarrow u] : C \to D}$$

   is turned into the following derivation:

$$\mathtt{lam}\,\frac{\mathtt{sub}\,\dfrac{\Gamma, y : C \vdash u : A \quad \Gamma, y : C, x : A \vdash t : D}{\Gamma, y : C \vdash t[x \leftarrow u] : D}}{\Gamma \vdash \lambda y.t[x \leftarrow u] : C \to D}$$

5. The reduction rule $(t\ v)[x \leftarrow u] \longrightarrow_{\mathtt{app}} t[x \leftarrow u]\ v[x \leftarrow u]$ corresponds to the permutation of the cut above an elimination, so that the derivation:

$$\mathtt{sub}\,\frac{\Gamma \vdash u : A \qquad \mathtt{app}\,\dfrac{\Gamma, x : A \vdash v : C \quad \Gamma, x : A \vdash t : C \to B}{\Gamma, x : A \vdash t\ v : B}}{\Gamma \vdash (t\ v)[x \leftarrow u] : B}$$

   is turned into the following derivation:

$$\mathtt{app}\,\frac{\mathtt{sub}\,\dfrac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash v : C}{\Gamma \vdash v[x \leftarrow u] : C} \qquad \mathtt{sub}\,\dfrac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : C \to B}{\Gamma \vdash t[x \leftarrow u] : C \to B}}{\Gamma \vdash t[x \leftarrow u]\ v[x \leftarrow u] : B}$$

   and the complete typing derivation above the premise $\Gamma \vdash u : A$ needs to be duplicated, and plugged above both copies of this premise.

In this analysis, we can see how the transformation of a detour into a cut instance corresponds to the B rule in $\lambda\mathtt{x}$, which triggers a substitution by creating the explicit substitution object in the term, and how the permutation of one cut corresponds to the other rules, where the explicit substitution is simply pushed through the term, to the variables.

Refined calculi. The correspondence established for $\mathsf{NJ} \cup \{\mathsf{cut}\}$ allows only to handle the $\lambda\mathtt{x}$-calculus, with its basic handling of erasure and duplication, but this can be extended, by modifying the proof system, to more refined λ-calculi with explicit substitutions. For example, we can use a separate weakening rule and allow the use of an axiom only when the antecedent contains only one formula. Since a cut then cannot reach an axiom on an unrelated formula, this would correspond to a λ-calculus with a form of *garbage collection*, where unused explicit substitutions can be erased before they reach a *» dead end «*. Similarly, separating the contraction from other inference rules implies a correspondence to a calculus where duplication is not performed only at the point where a substitution is propagated inside an application.

$$\mathtt{var}\ \frac{}{x:A \vdash x:A} \qquad \mathtt{sub}\ \frac{\Gamma \vdash u:A \quad \Delta, x:A \vdash t:B}{\Gamma, \Delta \vdash t[x \leftarrow u]:B}$$

$$\mathtt{rem}\ \frac{\Gamma \vdash t:B}{\Gamma, x:A \vdash t:B} \qquad \mathtt{dup}\ \frac{\Gamma, x:A, y:A \vdash t_{[y/x]}:B}{\Gamma, x:A \vdash t:B}$$

$$\mathtt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \mathtt{app}\ \frac{\Gamma \vdash u:A \quad \Delta \vdash t:A \to B}{\Gamma, \Delta \vdash t\,u:B}$$

Figure 5: Type system Ss for the λs-calculus

The variant of NJ where both weakening and contraction are separated from other rules can be used as type system for the λs-calculus, as illustrated by the type system Ss, shown in Figure 5. The dynamics of detour elimination is mostly the same as in NJ, but the handling of weakening and contraction yields the following rules of the λs-calculus:

1. The reduction rule $t[x \leftarrow u] \longrightarrow_{\mathtt{not}} t$, which can be applied when $x \notin t$, corresponds to the erasure of the cut when it encounters a weakening:

$$\mathtt{sub}\ \frac{\Gamma \vdash u:A \qquad \mathtt{rem}\ \dfrac{\Delta \vdash t:B}{\Delta, x:A \vdash t:B}}{\Gamma, \Delta \vdash t[x \leftarrow u]:B} \quad \longrightarrow \quad \mathtt{rem}^*\ \frac{\Delta \vdash t:B}{\Gamma, \Delta \vdash t:B}$$

   where rem instances must be added to erase substitutions corresponding to the free variables of u, which have disappeared.

2. The reduction rule $t[x \leftarrow u] \longrightarrow_{\mathtt{dup}} t_{[y/x]}[y \leftarrow u][x \leftarrow u]$ corresponds to the duplication of the cut when it encounters a contraction, so that the derivation:

$$\mathtt{sub}\ \frac{\Gamma \vdash u:A \qquad \mathtt{dup}\ \dfrac{\Delta, x:A, y:A \vdash t_{[y/x]}:B}{\Delta, x:A \vdash t:B}}{\Gamma, \Delta \vdash t[x \leftarrow u]:B}$$

   is turned into the following derivation:

$$\mathtt{sub}\ \frac{\Gamma \vdash u:A \qquad \mathtt{sub}\ \dfrac{\Gamma \vdash u:A \quad \Delta, x:A, y:A \vdash t_{[y/x]}:B}{\Gamma, \Delta, x:A \vdash t_{[y/x]}[y \leftarrow u]:B}}{\mathtt{dup}^*\ \dfrac{\Gamma, \Gamma, \Delta \vdash t_{[y/x]}[y \leftarrow u][x \leftarrow u]:B}{\Gamma, \Delta \vdash t_{[y/x]}[y \leftarrow u][x \leftarrow u]:B}}$$

   where new dup instances must be added to duplicate the environment Γ and have enough copies of other explicit substitutions to be propagated later into different copies of u.

The problem with the Ss type system is that both the rem and dup rules are not syntax-directed, and they can be applied at any time during the type inference process. This implies that there can be several valid typing derivations for a given λs-term, which is breaking the notion of full correspondence between intuitionistic proofs and λ-terms, as it exists for the natural deduction system NJ. Observe that this is related to the treatment of names in the duplication rule: depending on the use made of the rule dup when typing a given λs-term t, the rewriting step applied when the cut encounters the contraction is different — leading to different possible proofs, which are different possible typing derivations. Therefore, a unique typing derivation cannot model by its normalisation the reduction behaviour of a λs-term, this behaviour is described by a set of derivations. For example, the term $(\lambda y.t)[x \leftarrow u]$ can be typed by the two derivations:

$$\dfrac{\Gamma \vdash u : A \qquad \dfrac{\dfrac{\dfrac{t}{\Delta, x:A, z:A, y:B \vdash t_{[z/x]} : C}}{\Delta, x:A, z:A \vdash \lambda y.t_{[z/x]} : B \to C}\,\texttt{lam}}{\Delta, x:A \vdash \lambda y.t : B \to C}\,\texttt{dup}}{\Gamma, \Delta \vdash (\lambda y.t)[x \leftarrow u] : B \to C}\,\texttt{sub}$$

and

$$\dfrac{\Gamma \vdash u : A \qquad \dfrac{\dfrac{\dfrac{t}{\Delta, x:A, z:A, y:B \vdash t_{[z/x]} : C}}{\Delta, x:A, y:A \vdash t : C}\,\texttt{dup}}{\Delta, x:A \vdash \lambda y.t : B \to C}\,\texttt{lam}}{\Gamma, \Delta \vdash (\lambda y.t)[x \leftarrow u] : B \to C}\,\texttt{sub}$$

which correspond to two possible reductions of this term, where the duplication of the substitution is performed before or after pushing it through the abstraction:

$$\begin{aligned}(\lambda y.t)[x \leftarrow u] &\longrightarrow \lambda y.t[x \leftarrow u] \longrightarrow \lambda y.t_{[z/x]}[x \leftarrow u][z \leftarrow u] \\ &\longrightarrow (\lambda y.t_{[z/x]})[x \leftarrow u][z \leftarrow u] \longrightarrow^{*} \lambda y.t_{[z/x]}[x \leftarrow u][z \leftarrow u]\end{aligned}$$

so that these two derivations are part of the description of this term. The problem can be solved by introducing an explicit syntax for erasure and duplication, as done in the λlxr-calculus. In the syntax of this extension of λx, there are two resource operators, $\mathscr{W}_x$ for erasure and $\mathscr{C}_x^{z,w}$ for duplication. These two typing derivations, shown above, would correspond in λlxr to two distinct terms, $(\mathscr{C}_x^{z,w}.\lambda y.t)[x \leftarrow u]$ and $(\lambda y.\mathscr{C}_x^{z,w}.t)[x \leftarrow u]$. Reduction for each of these terms follows one of the two possible reductions shown above. This refined calculus is a more precise account of the computational contents of proofs in a variant of NJ where weakening and contraction are implemented in separate rules, but it is also more complex than λs, and has strong linearity constraints for terms to be well-formed. This is the price to pay for a representation of computation where resources are controlled than in the plain, standard λ-calculus and other of its extensions.

2.3 The λ-calculus with Pure Explicit Substitutions

The syntax of the traditional λ-calculus is ambiguous, in the sense that there is no way of knowing *a priori* wether an application $t\ u$ reduces to some redex $(\lambda x.v)\ u$ or to the simple application $x\ u$ of a function name x to a term u. Therefore, the usual syntax $t\ u$ assimilates two different realities, while the explicit substitutions syntax allows to distinguish β-redexes from other syntactic constructs. This yields a problem in the study of λ-calculi with explicit substitutions from the perspective of the Curry-Howard correspondence. Indeed, as mentioned before, the standard λ-calculi with explicit substitutions correspond to some natural deduction systems, where the cut rule from the sequent calculus has been artificially introduced. Only a few satisfactory correspondences have been established between sequent calculi and such λ-calculi, using some particular variants [Her94] or the standard system for intuitionistic logic [BG00, SU06]. This involves a generalisation of the shape of applications [JM00], and it often requires to define rules dealing with interactions between several applications [DP99]. In any case, the usual syntax for application must be dropped from the syntax, and the construction $t[x \leftarrow u]$ should be used as the only representation of a β-redex.

Moreover, the simplest case of application, of the shape $x\ u$, is not expressive enough to represent all possible applications in the standard λ-calculus. One could think of applying a variable to a list of arguments, but this is again not enough, as we may want to apply the result of a computation to an argument. To allow this kind of construction while keeping a separate syntax for applications and redexes, we can use a construct like $\texttt{let}\ x = u\ \texttt{in}\ t$, as presented in Moggi's *computational λ-calculus* [Mog89]. This syntax has the benefit of explicitly providing sequentiality information about applications in a flexible way, while with a term of the shape $(t\ v)\,(u\ w)$ one cannot add any information to indicate that $u\ w$ should be reduced first — this is interesting in the study of evaluation strategies, and in general in the λ-calculus because it is a theory of *sequential programs*. This also corresponds to a finer-grain representation of proofs, as found in the sequent calculus. We try here to keep the calculi as simple as possible, and for this reason we will not push the generalisation of application is it is done in some other studies of the computational contents of sequent calculi [San09].

We want a syntax using this sequential construct, but where the application of unrestricted terms is disallowed. This can be done by using only the particular case where a name is given to the result of applying a variable to a term, which can be written as $\texttt{let}\ x = z\ t\ \texttt{in}\ u$, where z is some variable[10].

Definition 2.18. *The language of* sequentialised and explicit λ*-terms is defined as:*

$$t, u\ ::=\ x \mid \lambda x.t \mid \texttt{let}\ x = z\ t\ \texttt{in}\ u \mid t[x \leftarrow u]$$

We use the same syntactic convention as in the standard calculi of the previous sections. In this syntax, the two constructs $\lambda x.t$ and $t[x \leftarrow u]$ bind x in t, and the construct $\texttt{let}\ x = z\ t\ \texttt{in}\ u$ binds x in u, not in t.

[10] This solution has been used in the literature to design calculi corresponding to the structure of the sequent calculus, but this is often not well-developped [SU06], and there is no real » *standard* « in this field, since these λ-calculi have rarely been studied outside of the typed setting.

We always consider terms *modulo* α-conversion, which is defined the same way as it was in the standard syntax, so that variables are bound at most once in a term. The sets of *bound variables* of the term t, denoted by $\mathrm{bv}(t)$, and of *free variables* of t, denoted by $\mathrm{fv}(t)$, are defined again as in the standard syntax, with a trivial adaptation to handle sequentialised application. We also reuse the notation $x \in t$ to indicate that a variable x appears free in the term t, and the notation $|t|_x$ for the number of occurrences of x that appear in t. The notion of substitution is also defined as before, without clashes of names thanks to the α-conversion convention.

We are again interested in *strong reduction*, so that we have to use a basic set of reduction rules in all the rewrite systems defined later in this section:

$$\begin{array}{rcll} \lambda x.t & \longrightarrow & \lambda x.v & \text{if } t \longrightarrow v \\ \texttt{let } x = z\ t\ \texttt{in}\ u & \longrightarrow & \texttt{let } x = z\ v\ \texttt{in}\ u & \text{if } t \longrightarrow v \\ \texttt{let } x = z\ t\ \texttt{in}\ u & \longrightarrow & \texttt{let } x = z\ t\ \texttt{in}\ v & \text{if } u \longrightarrow v \\ t[x \leftarrow u] & \longrightarrow & v[x \leftarrow u] & \text{if } t \longrightarrow v \\ t[x \leftarrow u] & \longrightarrow & t[x \leftarrow v] & \text{if } u \longrightarrow v \end{array} \tag{4}$$

In order to relate the sequentialised calculi that are presented in this section to the standard calculi described in the previous sections, we follow the same scheme, starting with a naïve approach to explicit substitutions, improving the calculus to refine it into a more interesting calculus, with a rich operational behaviour. Such a calculus, where the application is generalised into a sequential application that cannot be confused with a β-redex, will be called here a *pure explicit substitutions calculus*, because it focuses on the explicit substitution construct and removes the need for the B reduction rule, which does not apply on an explicit substitution.

Basic implementation. The first, naïve approach to pure explicit substitutions, that we call the $\lambda\overline{\mathtt{x}}$-calculus, is a variant of the standard $\lambda\mathtt{x}$-calculus integrating the syntax with sequentialised application. The set of reduction rules used in this system is presented in Figure 6, to which the rules of (4) are added to form the $\longrightarrow_{\lambda\overline{\mathtt{x}}}$ reduction — this is the basic approach in this setting [SU06].

Some of the rules are similar to those of the $\lambda\mathtt{x}$-calculus, and the `let` rule is an adaptation of the rule for application in the context of sequentialised application, where the substitution is pushed down on both sides of the application. Then, the `apd` reduction rule can be seen as a compound rule, necessary because there is no standard application syntax. It is equivalent to the following reduction sequence in a $\lambda\mathtt{x}$-like system that would allow arbitrary named applications:

$$\begin{array}{rl} & (\texttt{let } y = x\ v\ \texttt{in}\ t)[x \leftarrow \lambda z.u] \\ \longrightarrow & (\texttt{let } y = x[x \leftarrow \lambda z.u]\ v\ \texttt{in}\ t)[x \leftarrow \lambda z.u] \\ \longrightarrow & (\texttt{let } y = (\lambda z.u)\ v\ \texttt{in}\ t)[x \leftarrow \lambda z.u] \\ \longrightarrow & (\texttt{let } y = u[z \leftarrow v]\ \texttt{in}\ t)[x \leftarrow \lambda z.u] \\ \longrightarrow & t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u] \end{array}$$

but it is defined as one single-step reduction rule because there is no representation of the intermediate steps in the syntax of the $\lambda\overline{\mathtt{x}}$-calculus.

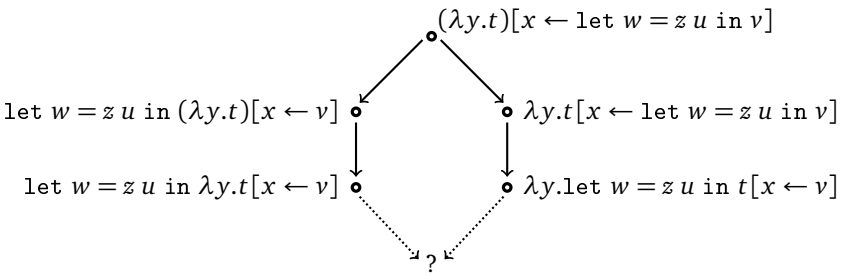
$$
\begin{array}{rcl}
t[x \leftarrow y] & \longrightarrow_{\texttt{ren}} & t\{y/x\} \\
x[x \leftarrow u] & \longrightarrow_{\texttt{var}} & u \\
z[x \leftarrow u] & \longrightarrow_{\texttt{nov}} & z \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\texttt{lam}} & \lambda y.t[x \leftarrow u] \\
(\texttt{let } y = z\ v \texttt{ in } t)[x \leftarrow u] & \longrightarrow_{\texttt{let}} & \texttt{let } y = z\ v[x \leftarrow u] \texttt{ in } t[x \leftarrow u] \\
(\texttt{let } y = x\ v \texttt{ in } t)[x \leftarrow \lambda z.u] & \longrightarrow_{\texttt{apd}} & t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u] \\
t_x[x \leftarrow \texttt{let } y = w\ v \texttt{ in } u] & \longrightarrow_{\texttt{out}} & \texttt{let } y = w\ v \texttt{ in } t[x \leftarrow u]
\end{array}
$$

where $z \neq x$ and t_x denotes a term of the shape $\texttt{let } z = x\ p \texttt{ in } q$

Figure 6: Reduction rules of the $\lambda\overline{x}$-calculus

This rule requires to get a λ-abstraction at toplevel inside the substitution, and this is only possible if we have rules to deal with other syntactic constructs that can appear. The first case left to treat is the case of an application, and this is handled through the `out` rule, which is unusual for a λ-calculus with explicit substitutions, although it needs to be introduced when using such a syntax [SU06, JM00]. Notice that the `out` rule is extremely constrained, so that the `let` construct can be moved out only at a particular position — this is meant to preserve confluence, so that the applications from the body of some substitution are not randomly located anywhere in the structure of the term after reduction. Finally, the case of a variable is treated through the `ren` rule, required since not all abstractions match an application.

Confluence. A problem of $\lambda\overline{x}$ is the limitation in moving applications imposed by the `out` rule, although one of the benefits of using this generalised application is a better control over the *localisation* of applications. We could therefore think of simplifying this rule and allow a `let` construct to move out of a substitution at any time, but this yields the following problematic situation:



where two different reduction sequences produce different sequential organisations of abstractions and applications. A solution could be the use of equations on terms, to allow for example the exchange of any unrelated abstractions and applications, so that the two reduced terms above would be considered equal. But this comes with some problems, as it would require to equate a term t where z does not appear free with any term of the shape $\texttt{let } z = x\ u \texttt{ in } t$. Moreover, this would reduce the interest of placing applications at particular locations.

In the setting of pure explicit substitutions, it is not obvious that a given calculus can simulate the standard λ-calculus, because the reduction rules operate in a quite different way. This is a good opportunity to illustrate how to prove confluence using the parallel reductions technique [Bar84] mentioned before, which is independent from the fact that a calculus can or cannot simulate β-reduction — as $\lambda\overline{x}$ is present but underdevelopped in the literature, there seem to be no confluence proof for it. This method is useful in the setting of pure explicit substitutions, since it is quite versatile and can be adapted to various calculi based on the same ideas as standard λ-calculi with or without explicit substitutions. For example, it can use a parallel reduction system where rules are grouped to produce a term normal with respect to a subsystem — often the group of rules pushing substitutions inside terms. Here, we will define a direct parallel variant of $\longrightarrow_{\lambda\overline{x}}$.

Definition 2.19. *The* parallel reduction *for $\lambda\overline{x}$ is denoted by $\Rightarrow_{\lambda\overline{x}}$ and defined as the reflexive closure of the the following set of reduction rules:*

$$
\begin{array}{rcl}
\lambda x.t & \Rightarrow & \lambda x.t_1 \\
\texttt{let } y = x\ t \texttt{ in } u & \Rightarrow & \texttt{let } y = x\ t_1 \texttt{ in } u_1 \\
t[x \leftarrow u] & \Rightarrow & t_1[x \leftarrow u_1] \\
t[x \leftarrow y] & \Rightarrow & t_1\{y/x\} \\
x[x \leftarrow u] & \Rightarrow & u_1 \\
z[x \leftarrow u] & \Rightarrow & z \\
(\lambda y.t)[x \leftarrow u] & \Rightarrow & \lambda y.t_1[x \leftarrow u_1] \\
(\texttt{let } y = z\ v \texttt{ in } t)[x \leftarrow u] & \Rightarrow & \texttt{let } y = z\ v_1[x \leftarrow u_1] \texttt{ in } t_1[x \leftarrow u_2] \\
(\texttt{let } y = x\ v \texttt{ in } t)[x \leftarrow \lambda z.u] & \Rightarrow & t_1[y \leftarrow u_1[z \leftarrow v_1]][x \leftarrow \lambda z.u_2] \\
t_x[x \leftarrow \texttt{let } y = z\ v \texttt{ in } u] & \Rightarrow & \texttt{let } y = z\ v_1 \texttt{ in } (\texttt{let } w = x\ p_1 \texttt{ in } q_1)[x \leftarrow u_1]
\end{array}
$$

where t_x is the term $\texttt{let } w = x\ p \texttt{ in } q$, and we have[11] *$t \Rightarrow t_1$ and $u \Rightarrow u_1$ as well as $u \Rightarrow u_2$ and $v \Rightarrow v_1$, and similarly $p \Rightarrow p_1$ and $q \Rightarrow q_1$, so that the definition is done inductively on the terms.*

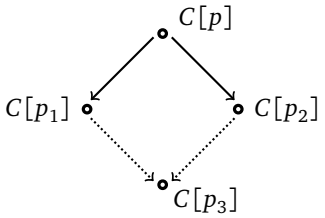
This system performs multiple one-step reductions on different redexes in the given term. The idea is that if two reductions can be applied on redexes that do not overlap, then they can be performed simultaneously, so that two reduction steps can be merged into one single compound reduction step, where every basic step corresponds to a step that can be performed in the basic $\longrightarrow_{\lambda x}$ reduction system. This allows us to prove that this parallel system has the diamond property, using a case analysis on the different rules that can be applied on a given term.

Lemma 2.20. *The $\Rightarrow_{\lambda\overline{x}}$ reduction has the diamond property.*

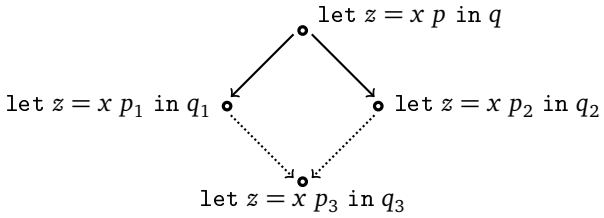
Proof. We proceed by induction on the size of a $\lambda\overline{x}$-term t, using a case analysis at each step on the reduction rules that can be applied to t. In the base case, t is some variable x and the result is trivial. Then, in the general case, we prove that given u and v such that $t \Rightarrow_{\lambda\overline{x}} u$ and $t \Rightarrow_{\lambda\overline{x}} v$ there is a term to which we can build two one-step reductions, to close the diagram.

[11] In the following, we will use a similar notation, where for example a term t reduces into t_1 and t_2, and the term t_1 might reduce into another term t_3, and so on.

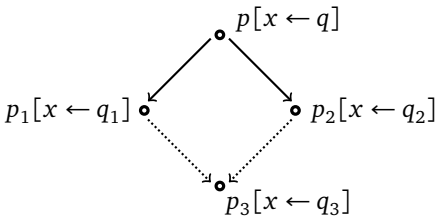
1. If t, u and v have the same toplevel structure, we use directly the induction hypothesis on the different subterms:



2. If u and v have both applications at toplevel, then we can use the induction hypothesis twice, on the corresponding subterms:

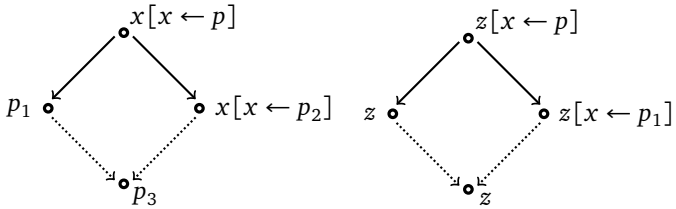


3. If u and v have both explicit substitutions at toplevel, we do the same and use the induction hypothesis twice, on the corresponding subterms:

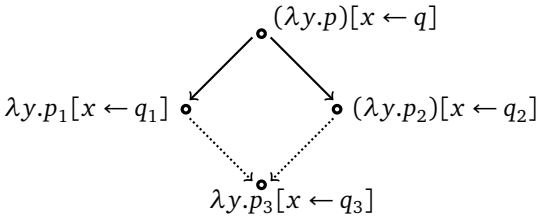


4. If t is of the shape $p[x \leftarrow y]$, we can use directly the induction hypothesis on the term p, and close the diagram immediatly, since the reduction step thus performed on p is not affected by the renaming of x into y.

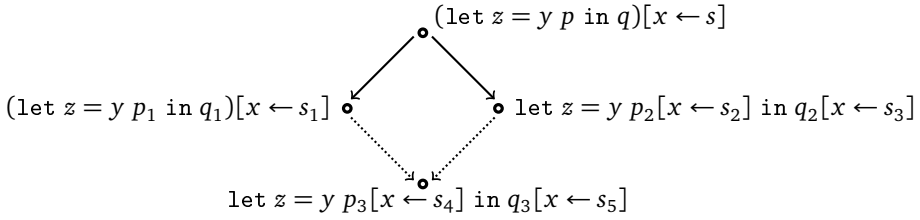
5. If t is of the shape $z[x \leftarrow p]$, we use the induction hypothesis on p, in both of the cases where z is x or a different variable, as follows:



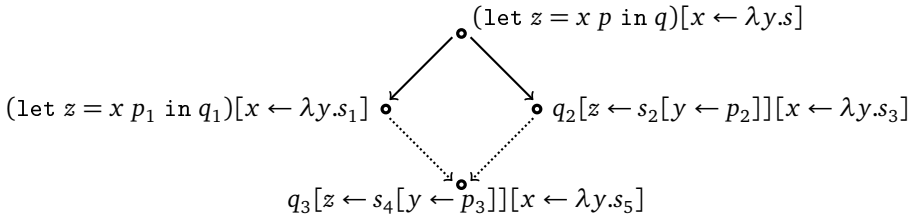
6. If t is of the shape $(\lambda y.p)[x \leftarrow q]$, we use the induction hypothesis on both p and q, and we can then build easily the resulting term:



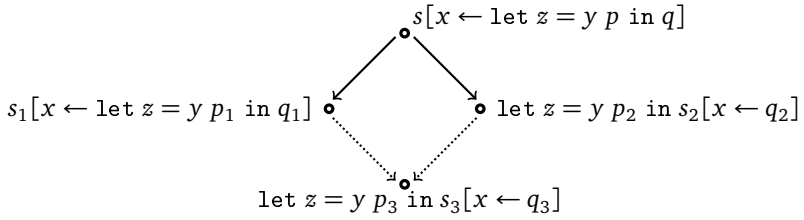
7. If t is of the shape $(\texttt{let}\ z = y\ p\ \texttt{in}\ q)[x \leftarrow s]$, we can again use directly the induction hypothesis on p, q and twice on s to build the resulting term:



8. If t is of the shape $(\texttt{let}\ z = x\ p\ \texttt{in}\ q)[x \leftarrow \lambda y.s]$, we can use the induction hypothesis on p, q and twice on s, as follows:



9. If t has the shape $s[x \leftarrow \texttt{let}\ z = y\ p\ \texttt{in}\ q]$ and s is $\texttt{let}\ w = x\ m\ \texttt{in}\ r$, we directly use the induction hypothesis on s, p and q to build the resulting term:



□

Now, we can prove that the calculus we are actually interested in is confluent, by showing how it has the same transitive closure as the parallel system.

Theorem 2.21. *The $\lambda\overline{x}$-calculus is confluent.*

Proof. First, we prove that the parallel system $\Rrightarrow_{\lambda\overline{x}}$ is confluent, which is immediate by applying Lemma 2.20, since the diamond property implies confluence, so that its closure $\Rrightarrow^*_{\lambda\overline{x}}$ is confluent. Then, we prove that $\Rrightarrow^*_{\lambda\overline{x}}$ and $\longrightarrow^*_{\lambda\overline{x}}$ are the same:

- If for two terms t and u we have $t \longrightarrow_{\lambda\overline{x}} u$ then it is clear that $t \Rrightarrow_{\lambda\overline{x}} u$, and thus we have the inclusion $\longrightarrow_{\lambda\overline{x}} \subseteq \Rrightarrow_{\lambda\overline{x}}$ which implies $\longrightarrow^*_{\lambda\overline{x}} \subseteq \Rrightarrow^*_{\lambda\overline{x}}$.
- If we have $t \Rrightarrow_{\lambda\overline{x}} u$ then by induction we can show that $t \longrightarrow^*_{\lambda\overline{x}} u$, and thus the inclusion $\Rrightarrow_{\lambda\overline{x}} \subseteq \longrightarrow^*_{\lambda\overline{x}}$, so that we also have $\Rrightarrow^*_{\lambda\overline{x}} \subseteq \longrightarrow^*_{\lambda\overline{x}}$.

Finally, since $\Rrightarrow^*_{\lambda\overline{x}}$ is confluent and is the same as the reduction $\longrightarrow^*_{\lambda\overline{x}}$ we know that $\longrightarrow^*_{\lambda\overline{x}}$ is confluent and therefore $\longrightarrow_{\lambda\overline{x}}$ is also confluent. □

Notice that this result would be slightly more complicated in a setting using equations rather than the restriction on the `out` rule. Indeed, a reduction diagram would then be closed by two terms which are considered equal, but it would also be necessary to consider a diagram formed by reductions on equal terms. This means that more reductions can be applied on a given term, since redexes can be created by associating subterms in other ways.

The most important problem of the $\lambda\overline{x}$-calculus is that it is weak, even weaker than the λx-calculus, since we cannot simulate β-reduction on any redex, and even the weak variant of full composition expressed by Lemma 2.6 does not hold. The reason is that there is no rule for composing or exchanging substitutions, although β-redexes must be translated as explicit substitutions. This means we cannot just choose any redex we want in a term and perform the associated substitution, since it needs to be in a particular position — it must be an *innermost* redex, involving a subterm where no other redex appears. General β-reduction, where no particular evaluation strategy is enforced, can thus not be simulated here. Because the ability to simulate one particular evaluation strategy is not enough for the goals of explicit substitutions calculi, we will not investigate further the properties of $\lambda\overline{x}$ and extend it to reach a decent[12] expressive power.

Remark 2.22. *Since the $\lambda\overline{x}$-calculus is too weak to simulate properly the standard λ-calculus, the PSN property is not interesting in this setting.*

Refined calculi. One way to extend the $\lambda\overline{x}$-calculus to a richer system is to add reduction rules for composing substitutions. However, as mentioned for standard calculi, this can induce non-terminating behaviours, and break the PSN property. We need to refine the $\lambda\overline{x}$-calculus into a new calculus similar to $\lambda\mathtt{es}$, where erasure and duplication are handled with care. The reduction rules for this calculus, called $\lambda\overline{e}$-calculus — which does not exist as such in the literature — are given in Figure 7, along with the usual equation on independent substitutions. Once again, the basic rules shown in (4) are implicitly part of the reduction system, and we will denote reduction in this calculus by $\longrightarrow_{\lambda\overline{e}}$ from now on.

[12] By » *decent expressive power* « we mean here that a calculus should at least have the full composition property, and thus allow stepwise simulation of β-reduction, to be really interesting — if it has no other purpose than implementing the standard λ-calculus, and no additional expressive feature.

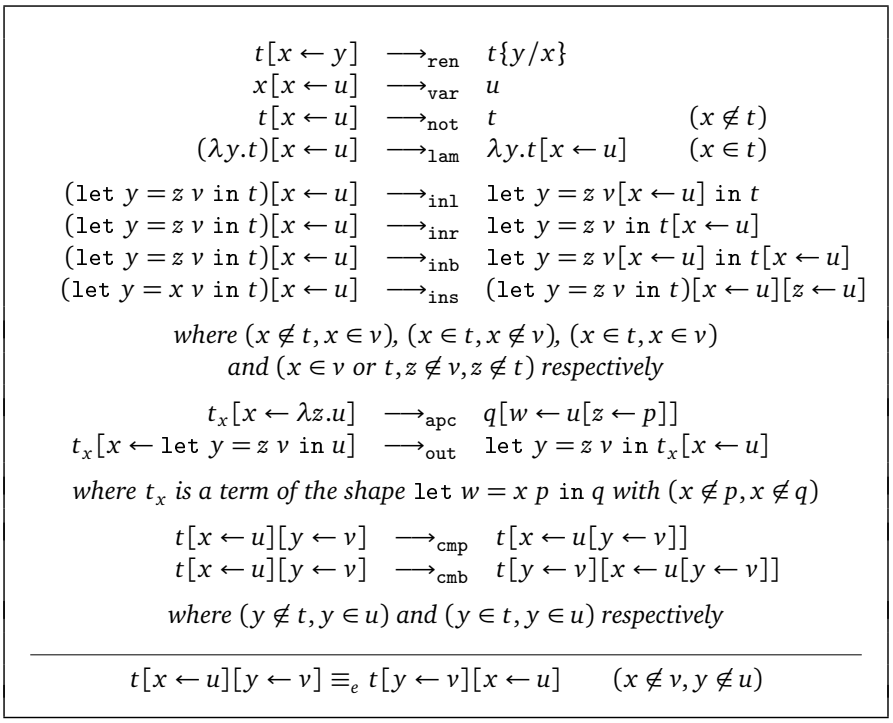


Figure 7: Reduction rules and equation of the $\lambda\overline{\mathtt{e}}$-calculus

This calculus is an adaptation of $\lambda\mathtt{es}$ to the sequentialised application setting, and it requires to add the $\equiv_e$ equation, as done in the $\lambda\mathtt{es}$-calculus, to handle cases where substitutions must be exchanged, without creating loops. However, in order to obtain a calculus that allows to simulate stepwise β-reduction, the handling of redexes done through the `apd` rule in $\lambda\overline{\mathtt{x}}$ has to be slightly complicated. The `apc` rule is actually a simplified variant, since it does not keep a copy of the substitution used, but the copy has to be performed separately by the `ins` rule, introducing a fresh name z for the variable being applied, so that the substitution can be pushed separately inside the subterms. For this reason, the variable x applied in both `apc` and `out` should not appear in the subterms.

The decomposition of the `apd` rule was not done in $\lambda\overline{\mathtt{x}}$ because the calculus would not have allowed for stepwise simulation of β-reduction even in this case, but also because it does not fit the naïve treatment of duplication of this setting. In the case of $\lambda\overline{\mathtt{e}}$, the use of the `ins` rule can be seen as a special case of copy.

Implicit substitution. The particular shape of the terms in the sequentialised syntax and the kind of reduction rules used in $\lambda\overline{\mathtt{e}}$ induce a more complex relation between explicit and implicit substitution than in the standard setting. Indeed, we need a specific definition of meta-level substitution to handle the lack of β-redexes, so that an explicit substitution can be created as the result of implicit substitution, at a particular position — directly applied on a `let` construct.

Definition 2.23. *The implicit substitution $t\{u/x\}$ of a term u for some variable x in another term t is defined in pure explicit substitutions calculi as:*

$$\begin{array}{rclcrcl} x\{u/x\} & = & u & & (\lambda y.t)\{u/x\} & = & \lambda y.t\{u/x\} \\ y\{u/x\} & = & y & & t[y \leftarrow v]\{u/x\} & = & t\{u/x\}[y \leftarrow v\{u/x\}] \end{array}$$

$$\begin{array}{rcl} (\mathtt{let}\ z = y\ v\ \mathtt{in}\ t)\{u/x\} & = & \mathtt{let}\ z = y\ v\{u/x\}\ \mathtt{in}\ t\{u/x\} \\ (\mathtt{let}\ z = x\ v\ \mathtt{in}\ t)\{u/x\} & = & (\mathtt{let}\ z = w\ v\{u/x\}\ \mathtt{in}\ t\{u/x\})[w \leftarrow u] \end{array}$$

where $y \neq x$ and w is a fresh variable, not free in u, v or t.

Relation to λ. The $\lambda\overline{\mathtt{e}}$-calculus is stronger than the $\lambda\overline{\mathtt{x}}$-calculus, since it allows to simulate a single step of β-reduction, and thus the general result of simulation of β-reduction, using the equation and reduction rules allowing to compose and exchange the substitutions. The starting point is the full composition result, which states that even with a different way of reducing terms — handling subterms inside and outside explicit substitutions alternatively — we can ensure a correspondence between explicit substitutions and meta-level, implicit substitutions. We denote by $\longrightarrow^{\equiv}_{\lambda\overline{\mathtt{e}}}$ the reduction *modulo* the congruence generated by the $\equiv_e$ equation, such that $t \longrightarrow^{\equiv}_{\lambda\overline{\mathtt{e}}} u$ when there are t' and u' such that $t \equiv t' \longrightarrow_{\lambda\overline{\mathtt{e}}} u' \equiv u$.

Lemma 2.24 (Full composition in $\lambda\overline{\mathtt{e}}$)**.** *For any t and u, $t[x \leftarrow u] \longrightarrow^{\equiv *}_{\lambda\overline{\mathtt{e}}} t\{u/x\}$.*

Proof. We proceed by structural induction on the given term t. In the base case, t is a variable and there are two possibilities. If t is $y \neq x$, we use the `not` rule, and if t is x, we use the `var` rule. In the general case, we use a case analysis on t:

1. If t is an abstraction $\lambda y.v$, we use the `lam` rule and the induction hypothesis on $v[x \leftarrow u]$ to reduce t to the expected term $\lambda y.v\{u/x\}$.
2. If t is some application $\mathtt{let}\ z = y\ v\ \mathtt{in}\ r$, we use the `inl`, `inr` or `inb` rule, depending on the use of x in the subterms v and r, and then use the induction hypothesis on $v[x \leftarrow u]$ or $r[x \leftarrow u]$, or both.
3. If t is an application $\mathtt{let}\ z = x\ v\ \mathtt{in}\ r$, there are two cases: if x does not appear in v or r then we are done, and if it does then we can use the `ins` rule and go on by induction hypothesis on $v[x \leftarrow u]$ and $r[x \leftarrow u]$.
4. If t is of the shape $v[y \leftarrow r]$, we use the `cmp` or `cmb` rule, or the $\equiv_e$ equation, depending on the use of x in the subterms v and r, and then use the induction hypothesis on $v[x \leftarrow u]$ or $r[x \leftarrow u]$, or both. □

We can use this result to prove that a single step of β-reduction can be simulated in the $\lambda\overline{\mathtt{e}}$-calculus, whatever strategy is used. Because this setting is different from more standard λ-calculi with explicit substitutions, we will need to define a specific translation from λ-terms into the $\lambda\overline{\mathtt{e}}$-calculus, relating the application schemes.

Definition 2.25. *The translation $[\![\cdot]\!]^{\lambda}_{\overline{\mathtt{e}}}$ from λ-terms to $\lambda\overline{\mathtt{e}}$-terms is defined as follows:*

$$\begin{array}{rcll} [\![x]\!]^{\lambda}_{\overline{\mathtt{e}}} & = & x & \\ [\![\lambda x.t]\!]^{\lambda}_{\overline{\mathtt{e}}} & = & \lambda x.[\![t]\!]^{\lambda}_{\overline{\mathtt{e}}} & \\ [\![t\ u]\!]^{\lambda}_{\overline{\mathtt{e}}} & = & (\mathtt{let}\ z = x\ [\![u]\!]^{\lambda}_{\overline{\mathtt{e}}}\ \mathtt{in}\ z)[x \leftarrow [\![t]\!]^{\lambda}_{\overline{\mathtt{e}}}] & \textit{(x and z fresh)} \end{array}$$

This translation is particular, as it introduces names to articulate the application of a term to another while respecting the scheme for application in $\lambda\overline{\mathsf{e}}$. One direct consequence of this is that the translation of a given λ-term in normal form is not always in normal form in $\lambda\overline{\mathsf{e}}$. However, we can prove that the reductions yet to be performed on such a term are reduced to a small subset of the rules.

Proposition 2.26. *For any λ-term t in normal form, there is a $\lambda\overline{\mathsf{e}}$-term u in normal form such that we have $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} \longrightarrow^{*}_{\{\mathtt{ren},\mathtt{out}\}} u$.*

Proof. We proceed by structural induction on t, using a case analysis on its shape, with a trivial base case when t is a variable x. If t is of the shape $\lambda x.p$, we can go on directly by induction hypothesis on p since $[\![\lambda x.p]\!]^{\lambda}_{\overline{\mathsf{e}}} = \lambda x.[\![p]\!]^{\lambda}_{\overline{\mathsf{e}}}$. Finally, if t is of the shape $x\ p_1 \cdots p_n$ then we use another induction, on n. If n is 1 we have:

$$[\![x\ p_1]\!]^{\lambda}_{\overline{\mathsf{e}}} \;=\; (\mathtt{let}\ z_1 = x_1\ [\![p_1]\!]^{\lambda}_{\overline{\mathsf{e}}}\ \mathtt{in}\ z_1)[x_1 \leftarrow x] \;\longrightarrow_{\mathtt{ren}}\; \mathtt{let}\ z_1 = x\ [\![p_1]\!]^{\lambda}_{\overline{\mathsf{e}}}\ \mathtt{in}\ z_1$$

and in the general case we have:

$$[\![x\ p_1 \cdots p_k]\!]^{\lambda}_{\overline{\mathsf{e}}} \;\longrightarrow^{*}_{\mathtt{ren},\mathtt{out}}\; \mathtt{let}\ z_1 = x\ [\![p_1]\!]^{\lambda}_{\overline{\mathsf{e}}}\ \mathtt{in}\ \cdots\ \mathtt{let}\ z_k = z_{k-1}\ [\![p_k]\!]^{\lambda}_{\overline{\mathsf{e}}}\ \mathtt{in}\ z_k$$

so that after $k+1$ step, we can add k times the `out` rule to the reduction sequence to obtain the result for $[\![x\ p_1 \cdots p_{k+1}]\!]^{\lambda}_{\overline{\mathsf{e}}}$, which can obviously be turned in normal form if all the $[\![p_i]\!]^{\lambda}_{\overline{\mathsf{e}}}$ can be turned into normal forms as well following the same procedure — and this is ensured by the induction hypothesis. □

This translation is thus acceptable, as the `ren` and `out` rule implement minor rewritings, mostly reordering sequences of applications. Also notice that sequential applications in the translation of a term are all of the shape $\mathtt{let}\ z = x\ v\ \mathtt{in}\ z$. We can now use the full composition lemma and the translation to prove the stepwise simulation of β-reduction, based on the observation that the implicit substitution is compatible with this translation.

Lemma 2.27. *For any $\lambda\overline{\mathsf{e}}$-terms t and u we have $[\![t\{u/x\}]\!]^{\lambda}_{\overline{\mathsf{e}}} = [\![t]\!]^{\lambda}_{\overline{\mathsf{e}}}\{[\![u]\!]^{\lambda}_{\overline{\mathsf{e}}}/x\}$.*

Proof. We proceed by structural induction on t. First, if t is y then the substitution has no effect, and if t is x then $t\{u/x\}$ is u and the translation is indeed $[\![u]\!]^{\lambda}_{\overline{\mathsf{e}}}$. If t is $\lambda y.p$, then $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} = \lambda y.[\![p]\!]^{\lambda}_{\overline{\mathsf{e}}}$ and we use the induction hypothesis on p. Finally, if t is of the shape $p\ q$ then $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} = (\mathtt{let}\ z = y\ [\![q]\!]^{\lambda}_{\overline{\mathsf{e}}}\ \mathtt{in}\ z)[y \leftarrow [\![p]\!]^{\lambda}_{\overline{\mathsf{e}}}]$ and by definition of the substitution, we can use the induction hypothesis on p and q. □

Theorem 2.28 (Simulation in $\lambda\overline{\mathsf{e}}$)**.** *For t and u, if $t \longrightarrow_{\beta} u$ then $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} \longrightarrow^{\equiv *}_{\lambda\overline{\mathsf{e}}} [\![u]\!]^{\lambda}_{\overline{\mathsf{e}}}$.*

Proof. We proceed by structural induction on t. If t is $\lambda x.p$, we use the induction hypothesis on p, since if $[\![p]\!]^{\lambda}_{\overline{\mathsf{e}}} \longrightarrow^{\equiv *}_{\overline{\mathsf{e}}} [\![q]\!]^{\lambda}_{\overline{\mathsf{e}}}$ then $\lambda x.[\![p]\!]^{\lambda}_{\overline{\mathsf{e}}} \longrightarrow^{\equiv *}_{\overline{\mathsf{e}}} \lambda x.[\![q]\!]^{\lambda}_{\overline{\mathsf{e}}}$. If t is $p\ q$ and the reduction happens inside p or inside q, we use the induction hypothesis on p or q respectively, for the same reason as in the previous case. Finally, if t is $(\lambda x.p)\ q$ and u is $p\{q/x\}$, then $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} = (\mathtt{let}\ z = y\ [\![q]\!]^{\lambda}_{\overline{\mathsf{e}}}\ \mathtt{in}\ z)[y \leftarrow \lambda x.[\![p]\!]^{\lambda}_{\overline{\mathsf{e}}}]$ and we have $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} \longrightarrow_{\mathtt{apc},\mathtt{var}} [\![p]\!]^{\lambda}_{\overline{\mathsf{e}}}[x \leftarrow [\![q]\!]^{\lambda}_{\overline{\mathsf{e}}}] \longrightarrow^{\equiv *}_{\lambda\overline{\mathsf{e}}} [\![p]\!]^{\lambda}_{\overline{\mathsf{e}}}\{[\![q]\!]^{\lambda}_{\overline{\mathsf{e}}}/x\}$ by Lemma 2.24, so that $[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}} \longrightarrow^{\equiv *}_{\lambda\overline{\mathsf{e}}} [\![p\{q/x\}]\!]^{\lambda}_{\overline{\mathsf{e}}}$ through the result of Lemma 2.27. □

In order to also prove the projection result, which ensures that this simulation is meaningful, in the sense that it creates a close correspondence between reduction in $\lambda\overline{\mathsf{e}}$ and the λ-calculus, we need to define the opposite translation, shown below and based on the transformation of `let` constructs into standard applications.

Definition 2.29. *The translation $[\![\cdot]\!]^{\overline{\mathsf{e}}}_{\lambda}$ from $\lambda\overline{\mathsf{e}}$-terms to λ-terms is defined as follows:*

$$\begin{array}{rclcrcl} [\![x]\!]^{\overline{\mathsf{e}}}_{\lambda} & = & x & \qquad & [\![\mathtt{let}\ z = x\ u\ \mathtt{in}\ t]\!]^{\overline{\mathsf{e}}}_{\lambda} & = & [\![t]\!]^{\overline{\mathsf{e}}}_{\lambda}\{x\ [\![u]\!]^{\overline{\mathsf{e}}}_{\lambda}/z\} \\ [\![\lambda x.t]\!]^{\overline{\mathsf{e}}}_{\lambda} & = & \lambda x.[\![t]\!]^{\overline{\mathsf{e}}}_{\lambda} & & [\![t[x \leftarrow u]]\!]^{\overline{\mathsf{e}}}_{\lambda} & = & [\![t]\!]^{\overline{\mathsf{e}}}_{\lambda}\{[\![u]\!]^{\overline{\mathsf{e}}}_{\lambda}/x\} \end{array}$$

Theorem 2.30 (Projection in $\lambda\overline{\mathsf{e}}$)**.** *For t and u, if $t \longrightarrow_{\lambda\overline{\mathsf{e}}} u$ then $[\![t]\!]^{\overline{\mathsf{e}}}_{\lambda} \longrightarrow^{*}_{\beta} [\![u]\!]^{\overline{\mathsf{e}}}_{\lambda}$.*

Proof. By case analysis on the rule used to reduce t into u. Because of contextual closure, and of compositionality of the translation $[\![\cdot]\!]^{\overline{\mathsf{e}}}_{\lambda}$, we assume without loss of generality that the rule is applied at the root of t.

1. If one of the rules `ren`, `var` and `not` was used, then it is clear, by definition of the implicit substitution operation, that t and u have the same translation.
2. If one of the rules `inl`, `inr`, `inb` and `ins` was used, then t and u again have the same translation, by definition of the translation, and possibly using the standard substitution lemma [Klo80] of the λ-calculus. The situation is the same if one of the rules `lam`, `out`, `cmp` and `cmb` was used.
3. If the rule `apc` was used, then t is of the shape $(\mathtt{let}\ z = x\ v\ \mathtt{in}\ r)[x \leftarrow \lambda y.s]$ and thus $[\![t]\!]^{\overline{\mathsf{e}}}_{\lambda}$ is $[\![r]\!]^{\overline{\mathsf{e}}}_{\lambda}\{x\ [\![v]\!]^{\overline{\mathsf{e}}}_{\lambda}/z\}\{\lambda y.[\![s]\!]^{\overline{\mathsf{e}}}_{\lambda}/x\}$, which can be reduced by β into $[\![r]\!]^{\overline{\mathsf{e}}}_{\lambda}\{[\![s]\!]^{\overline{\mathsf{e}}}_{\lambda}\{[\![v]\!]^{\overline{\mathsf{e}}}_{\lambda}/y\}/z\}$, and this is exactly $[\![u]\!]^{\overline{\mathsf{e}}}_{\lambda}$.

This means that one application of the `apc` rule can be projected as one β-reduction, and all other rules preserve the translation of the initial term, so that reduction in $\lambda\overline{\mathsf{e}}$ can indeed be projected into the standard β-reduction system. □

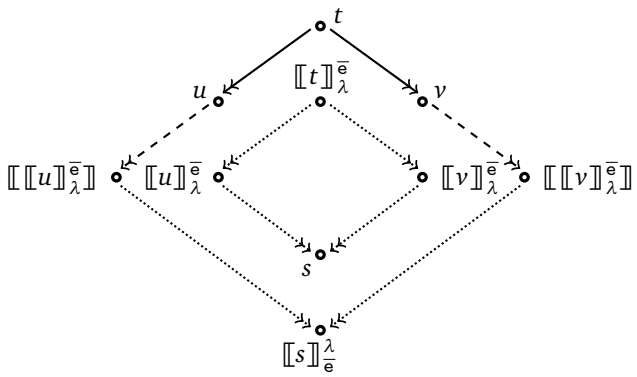
Notice that the translation used for the projection result allows to retrieve some original λ-term t from its translation into $\lambda\overline{\mathsf{e}}$, because it collapses the structures of explicit substitutions and sequential applications into the correct applications, in the special case where these have been created by the translation $[\![\cdot]\!]^{\lambda}_{\overline{\mathsf{e}}}$. Precisely, we have for any λ-term t the equation $[\![[\![t]\!]^{\lambda}_{\overline{\mathsf{e}}}]\!]^{\overline{\mathsf{e}}}_{\lambda} = t$, as illustrated in the following example:

$$[\![[\![p\ q]\!]^{\lambda}_{\overline{\mathsf{e}}}]\!]^{\overline{\mathsf{e}}}_{\lambda} = [\![(\mathtt{let}\ z = x\ q\ \mathtt{in}\ z)[x \leftarrow p]]\!]^{\overline{\mathsf{e}}}_{\lambda} = z\{x\ [\![q]\!]^{\overline{\mathsf{e}}}_{\lambda}/z\}\{[\![p]\!]^{\overline{\mathsf{e}}}_{\lambda}/x\} = p\ q$$

but the equation does not hold if translations are inverted. Indeed, the structure of the pure λ-calculus is too weak to preserve the order of the sequential applications, and the translation from $\lambda\overline{\mathsf{e}}$ cannot be used to retrieve the original term.

Confluence. Now that we have established a strong correspondence between reduction in $\lambda\overline{\mathsf{e}}$ and in the standard λ-calculus, we can apply the usual techniques to investigate the operational properties of $\lambda\overline{\mathsf{e}}$, starting with confluence. However, we cannot use full composition for this, as we would need for this a lemma similar to Proposition 2.10, but standard λ-terms are not a particular kind of $\lambda\overline{\mathsf{e}}$-terms.

A solution following this idea would thus require a translation back from the λ-calculus, but the translation into $\lambda\overline{\mathtt{e}}$ of the λ-calculus translation of a term is not the same term, and we cannot reduce one into the other[13], so that we *cannot* use the following diagram with $[\![\cdot]\!]$ defined as $[\![\cdot]\!]^{\overline{\mathtt{e}}}_{\lambda}$:



and we will not define another translation that would allow to prove simulation and projection, and interact nicely with the translation $[\![\cdot]\!]^{\overline{\mathtt{e}}}_{\lambda}$. Fortunately, there are other methods to overcome this problem.

Since we are in a situation similar to the one of $\lambda\overline{\mathtt{x}}$, we apply the same method and use parallel reductions. In order to simplify the definition of this reduction, we use normal forms, but in the setting of pure explicit substitutions, it is impossible to use normal forms without explicit substitutions [Kes07], since this would require termination. The solution is to isolate problematic rules, in particular the `apc` rule which corresponds to the standard B rule.

Definition 2.31. *The* pushing normal form *of a term t, denoted by $\mathtt{ps}(t)$, is a term obtained by maximal application of any rule of $\lambda\overline{\mathtt{e}}$ except* `apc` *and* `out`.

This normal form is well-defined because the part of the reduction $\longrightarrow^{\equiv}_{\lambda\overline{\mathtt{e}}}$ formed by these *pushing rules*, and called $\longrightarrow^{\equiv}_{\overline{\mathtt{e}}}$, can be shown to be terminating. For that, we need a measure that will decrease during reduction, adapted from $\lambda\mathtt{es}$ [Kes07].

Definition 2.32. *The* substitution rank *of a term t, denoted by $\mathtt{R}(t)$, is defined as:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{R}(x) & = & 1 & \mathtt{R}(\mathtt{let}\ y = z\ t\ \mathtt{in}\ u) & = & \mathtt{R}(t) + \mathtt{R}(u) + 1 \\
\mathtt{R}(\lambda x.t) & = & \mathtt{R}(t) & \mathtt{R}(t[x \leftarrow u]) & = & \mathtt{R}(t) + \mathtt{M}_x(t) \times (|t|^2_x + 1) \times \mathtt{R}(u)
\end{array}$$

$$\begin{array}{rcll}
\textit{where}\quad \mathtt{M}_x(z) & = & 1 & \textit{for any } z \\
\mathtt{M}_x(\lambda y.t) & = & \mathtt{M}_x(t) + 1 & \\
\mathtt{M}_x(\mathtt{let}\ y = z\ t\ \mathtt{in}\ u) & = & \mathtt{M}_x(t) + \mathtt{M}_x(u) + 1 & \\
\mathtt{M}_x(t[y \leftarrow u]) & = & \mathtt{M}_x(t) & \textit{if } x \notin u \\
\mathtt{M}_x(t[y \leftarrow u]) & = & \mathtt{M}_x(t) + \mathtt{M}_y(t) \times (|t|^2_y + 1) \times (\mathtt{M}_x(u) + 1) & \textit{if } x \in u
\end{array}$$

[13] This would require to move applications within a term, so that it would be possible in the system based on equations mentioned before, but not in the variant we use here.

Lemma 2.33. *For any t and u, if $t \longrightarrow^{\equiv}_{\overline{\mathtt{e}}} u$ then $\mathtt{R}(u) < \mathtt{R}(t)$.*

Proof. By case analysis on the reduction rule used for rewriting t into u, using the observation that for any v and any x, we have $\mathtt{R}(v) \geq 1$ and $\mathtt{M}_x(v) \geq 1$, and also the fact that if $v \equiv v'$ then we have $\mathtt{R}(v) = \mathtt{R}(v')$. □

Moreover, given a term t, the term $\mathtt{ps}(t)$ is uniquely defined, since this pushing subsystem is clearly confluent, as we always have $t[x \leftarrow u] \longrightarrow^{\equiv *}_{\overline{\mathtt{e}}} t\{u/x\}$. Notice that the equation $\equiv_e$ never associates two different terms in pushing normal form, because the only explicit substitutions left in such a term are located *directly* above the applications involving their bound variable. We need to prove two lemmas that describe how pushing normal forms interact with reduction and the $\equiv$ congruence.

Lemma 2.34. *For any terms t and u, if $t \longrightarrow^{*}_{\lambda\overline{\mathtt{e}}} u$ then $\mathtt{ps}(t) \longrightarrow^{\equiv *}_{\lambda\overline{\mathtt{e}}} \mathtt{ps}(u)$.*

Proof. We proceed by induction on the length of the reduction from t to u, and in the base case we have $t = u$ and thus $\mathtt{ps}(t) = \mathtt{ps}(u)$. In general, there is a v such that $t \longrightarrow_{\lambda\overline{\mathtt{e}}} v \longrightarrow^{*}_{\lambda\overline{\mathtt{e}}} u$ and we use a case analysis on the reduction from t to v. If v is obtained from t by a pushing rule, then $\mathtt{ps}(t) = \mathtt{ps}(v)$ and we can conclude by induction hypothesis on $v \longrightarrow^{*}_{\lambda\overline{\mathtt{e}}} u$. Otherwise, if $t \longrightarrow_{\mathtt{out}} v$ we have immediately $\mathtt{ps}(t) \longrightarrow_{\mathtt{out}} \mathtt{ps}(v)$ since $\mathtt{out}$ cannot create new redexes for pushing rules. Finally, if $t \longrightarrow_{\mathtt{apc}} v$ there are new substitutions inside v to push, we have some term s such that $\mathtt{ps}(t) \longrightarrow_{\mathtt{apc}} s \longrightarrow^{\equiv *}_{\overline{\mathtt{e}}} \mathtt{ps}(v)$. In both of these cases, we conclude by induction hypothesis on $v \longrightarrow^{*}_{\lambda\overline{\mathtt{e}}} u$. □

Lemma 2.35. *For any terms t and u, if $t \equiv u$ then $\mathtt{ps}(t) = \mathtt{ps}(u)$.*

Proof. This is a direct consequence of the confluence of the pushing subsystem, as a substitution inside t and u is pushed and either carried out completely or blocked above applications involving the variable it is binding, independently of the original location of the substitution in the term. □

Now, we can define the parallel reduction for $\lambda\overline{\mathtt{e}}$ on pushing normal forms, so that it hides the details of the pushing subsystem.

Definition 2.36. *The* parallel reduction *for $\lambda\overline{\mathtt{e}}$ is denoted by $\Rrightarrow_{\lambda\overline{\mathtt{e}}}$ and defined on pushing normal forms as the reflexive closure of the following set of rules:*

$$\begin{array}{rcl}
\lambda x.t & \Rrightarrow & \lambda x.t_1 \\
\mathtt{let}\ x = z\ u\ \mathtt{in}\ t & \Rrightarrow & \mathtt{let}\ x = z\ u_1\ \mathtt{in}\ t_1 \\
t[x \leftarrow u] & \Rrightarrow & t_1[x \leftarrow u_1] \\
t_x[x \leftarrow \lambda y.u] & \Rrightarrow & \mathtt{ps}(r_1[z \leftarrow u_1[y \leftarrow p_1]]) \\
t_x[x \leftarrow \mathtt{let}\ y = w\ q\ \mathtt{in}\ u] & \Rrightarrow & \mathtt{let}\ y = w\ q_1\ \mathtt{in}\ (\mathtt{let}\ z = x\ p_1\ \mathtt{in}\ r_1)[x \leftarrow u_1]
\end{array}$$

where t_x is the term $\mathtt{let}\ z = x\ p\ \mathtt{in}\ r$, with $(x \notin p, x \notin r)$, and we have $t \Rrightarrow t_1$ and $u \Rrightarrow u_1$ as well as $p \Rrightarrow p_1$ and $q \Rrightarrow q_1$ and $r \Rrightarrow r_1$.

Notice that the parallel reduction $\Rrightarrow_{\lambda\overline{\mathtt{e}}}$ is defined inductively, so that we may use an induction on the structure of a reduction $t \Rrightarrow_{\lambda\overline{\mathtt{e}}} u$, as in the following lemma.

Lemma 2.37. *For any terms t and u in pushing normal form, if we have $t \Rrightarrow_{\lambda\overline{\mathtt{e}}} t_1$ and $u \Rrightarrow_{\lambda\overline{\mathtt{e}}} u_1$ then we also have $\mathtt{ps}(t[x \leftarrow u]) \Rrightarrow_{\lambda\overline{\mathtt{e}}} \mathtt{ps}(t_1[x \leftarrow u_1])$.*

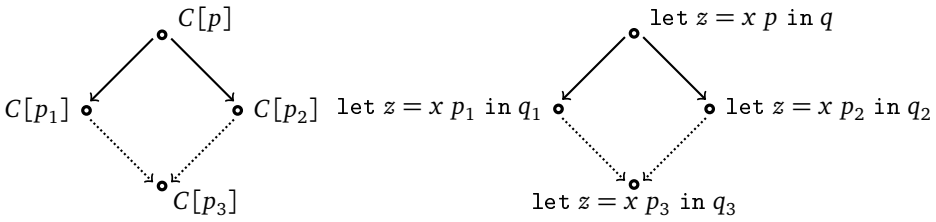
Proof. By induction on the structure of the reduction $t \Rrightarrow_{\lambda\overline{\mathtt{e}}} t_1$. In the base case we have $t = t_1$ and the result is immediate, since pushing a substitution inside some term is independent from the body of this substitution. If the reduction modifies only subterms of t, we use the induction hypothesis on these subterms, since the substitution is pushed the same way in t and t_1. If t is a redex for the `out` rule and an application is moved out of it, we can also use the induction hypothesis since the substitution can be pushed inside this application and inside other substitutions. Finally, if t is a redex for `apc`, the induction hypothesis also applies, since the new substitution created binds a variable that cannot appear in u. □

Then, we can prove that this parallel reduction has the diamond property, by a case analysis of a given term and of the possible reductions.

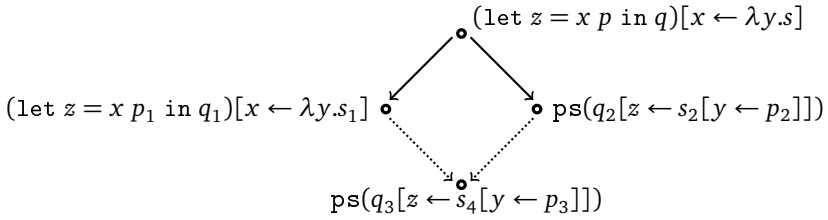
Lemma 2.38. *The $\Rrightarrow_{\lambda\overline{\mathtt{e}}}$ reduction has the diamond property.*

Proof. We proceed by induction on the size of a $\lambda\overline{\mathtt{e}}$-term t in pushing normal form, using a case analysis at each step on the reductions that can be applied to t. In the base case, t is some variable x and the result is trivial. Then, in the general case, we prove that given u and v such that $t \Rrightarrow_{\lambda\overline{\mathtt{e}}} u$ and $t \Rrightarrow_{\lambda\overline{\mathtt{e}}} v$ there is a term to which we can build two one-step reductions, to close the diagram:

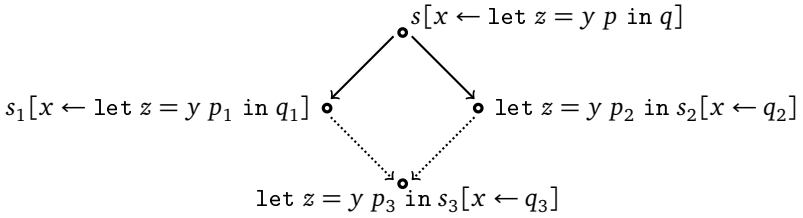
1. If t, u and v have the same toplevel structure, we use directly the induction hypothesis on the different subterms, and if u and v have both applications, or both substitutions, at toplevel, we can use the induction hypothesis twice, on the corresponding subterms, as shown below in the case of an application:



2. If t is of the shape $(\mathtt{let}\ z = x\ p\ \mathtt{in}\ q)[x \leftarrow \lambda y.s]$, we can use the induction hypothesis on p, q and s, as follows, and conclude by Lemma 2.37:



3. If t has the shape $s[x \leftarrow \texttt{let}\ z = y\ p\ \texttt{in}\ q]$ and s is $\texttt{let}\ w = x\ m\ \texttt{in}\ r$, we directly use the induction hypothesis on s, p and q to build the resulting term:



□

This allows to conclude that the basic reduction system for $\lambda\overline{\texttt{e}}$ is confluent, through its correspondence to the parallel reduction system.

Theorem 2.39. *The $\lambda\overline{\texttt{e}}$-calculus is confluent.*

Proof. We consider terms t and t' such that $t \equiv t'$ and $t \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} u$ and $t' \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} v$, to show that both u and v can be reduced into a unique term r. First observe that we have $u \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} \texttt{ps}(u)$ and $v \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} \texttt{ps}(v)$ by definition. Then, by Lemma 2.35 we know that $\texttt{ps}(t) = \texttt{ps}(t')$ and by Lemma 2.34 we conclude that $\texttt{ps}(t) \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} \texttt{ps}(u)$ and $\texttt{ps}(t) \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} \texttt{ps}(v)$. Moreover, we have:

- If for two terms t and u in pushing normal form we have $t \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} u$ then by induction on this reduction we can show that $t \Rightarrow^{*}_{\lambda\overline{\texttt{e}}} u$.
- If we have $t \Rightarrow_{\lambda\overline{\texttt{e}}} u$ then by induction on the structure of the reduction we can show that $t \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} u$, so that by repeating the operation we can also conclude that if $t \Rightarrow^{*}_{\lambda\overline{\texttt{e}}} u$ then we have $t \longrightarrow^{\equiv *}_{\lambda\overline{\texttt{e}}} u$ as well.

This implies that we can translate reduction sequences into the parallel reduction system, and thus we have $\texttt{ps}(t) \Rightarrow^{*}_{\lambda\overline{\texttt{e}}} \texttt{ps}(u)$ and $\texttt{ps}(t) \Rightarrow^{*}_{\lambda\overline{\texttt{e}}} \texttt{ps}(v)$. But the parallel reduction has the diamond property and therefore is confluent, so that there exists some term r such that $\texttt{ps}(u) \Rightarrow^{*}_{\lambda\overline{\texttt{e}}} r$ and $\texttt{ps}(v) \Rightarrow^{*}_{\lambda\overline{\texttt{e}}} r$, which provides the expected result, through the reductions from u and v to their pushing normal form. □

PSN. As in other λ-calculi of explicit substitutions, we may want to ensure that the translation from the standard λ-calculus into this refined $\lambda\overline{\texttt{e}}$-calculus preserves the most important property of a term, its normalisability. However, the situation here is more complex than usual, because the translation given in Definition 2.25 performs a significant amount of reorganisation within the term. In particular, the translation of standard applications into sequential ones changes the order in which arguments are encountered in the syntactic tree of the term, and new variables are introduced to represent intermediate results.

It seems reasonable to attempt proving the PSN property for $\lambda\overline{\texttt{e}}$ by reducing it to the PSN property of the standard $\lambda\texttt{es}$-calculus, since both calculi use the same mechanisms for handling the distribution of the explicit substitutions. But relating reduction steps in $\lambda\overline{\texttt{e}}$ to the steps of $\lambda\texttt{es}$ is made difficult by the particular shape of the apc rule, which corresponds to the transformation of a β-redex into a new explicit substitution — as this rule modifies an application in a way that cannot be

$$
\begin{array}{rcll}
t[x \leftarrow y] & \longrightarrow_{\mathtt{ren}} & t\{y/x\} & (|t|_x \geq 1) \\
x[x \leftarrow u] & \longrightarrow_{\mathtt{var}} & u & \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{not}} & t & (|t|_x = 0) \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] & (|t|_x \geq 1) \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{dup}} & t_{[y/x]}[x \leftarrow u][y \leftarrow u] & (|t|_x \geq 2)
\end{array}
$$

$$
\begin{array}{rcl}
(\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u] & \longrightarrow_{\mathtt{inl}} & \mathtt{let}\ y = z\ v[x \leftarrow u]\ \mathtt{in}\ t \\
(\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u] & \longrightarrow_{\mathtt{inr}} & \mathtt{let}\ y = z\ v\ \mathtt{in}\ t[x \leftarrow u]
\end{array}
$$

where $(|t|_x = 0, |v|_x \geq 1)$ *and* $(|t|_x \geq 1, |v|_x = 0)$ *respectively*

$$
\begin{array}{rcl}
t_x[x \leftarrow \lambda z.u] & \longrightarrow_{\mathtt{apc}} & p[y \leftarrow u[z \leftarrow v]] \\
t_x[x \leftarrow \mathtt{let}\ w = z\ q\ \mathtt{in}\ u] & \longrightarrow_{\mathtt{out}} & \mathtt{let}\ w = z\ q\ \mathtt{in}\ t_x[x \leftarrow u]
\end{array}
$$

where t_x *is of the shape* $\mathtt{let}\ y = x\ v\ \mathtt{in}\ p$ *with* $(|v|_x = 0, |p|_x = 0)$

$$
t[x \leftarrow u][y \leftarrow v] \longrightarrow_{\mathtt{cmp}} t[x \leftarrow u[y \leftarrow v]] \qquad (|t|_y = 0, |u|_y \geq 1)
$$

$$
t[x \leftarrow u][y \leftarrow v] \equiv_e t[y \leftarrow v][x \leftarrow u] \qquad (x \notin v, y \notin u)
$$

Figure 8: Reduction rules and equation of the $\lambda\overline{\mathtt{s}}$-calculus

immediately related to the structure of a corresponding standard application. The choice of reduction rules as adaptations of the rules of $\lambda\mathtt{es}$ into the setting of pure explicit substitutions, and the fact that $\mathtt{apc}$ was introduced as a compound of more standard steps, suggests that $\lambda\overline{\mathtt{e}}$ enjoys the PSN property, but we leave the question of the proof technique to use open.

Proving this result would show that explicit substitutions calculi, with rules for composition of substitutions, can be as well-behaved in the setting of pure explicit substitutions, with a sequential form of application, as in the standard case. Indeed, the $\lambda\overline{\mathtt{e}}$-calculus that we have presented here is confluent, it allows to simulate in a sensible way the standard λ-calculus, and with this result we would make sure that it preserves the strong normalisation of λ-terms, although it introduces a clear distinction between pure application and β-redexes.

Other calculi. As in the standard case, there are other possible presentations of a calculus with pure explicit substitutions, and in particular we can refine the $\lambda\overline{\mathtt{e}}$-calculus the same way $\lambda\mathtt{es}$ was refined into the $\lambda\mathtt{s}$-calculus. This would lead to the $\lambda\overline{\mathtt{s}}$-calculus, for which the reduction rules are shown above in Figure 8. The operational behaviour of this system is similar to the one of $\lambda\overline{\mathtt{e}}$, with the difference that duplication of substitutions is decoupled from propagation into terms that have several compound subterms, and side conditions are expressed in terms of the number of occurrences of variables bound in explicit substitutions. We could use the same techniques as before to show that this refined calculus is also confluent, simulates β-reduction and has the PSN property. Finally, the $\lambda\mathtt{lxr}$-calculus could also be adapted to the setting of pure explicit substitutions, by modifying $\lambda\overline{\mathtt{s}}$ for example, to control the $\mathtt{not}$ and $\mathtt{dup}$ rules with explicit resource operators.

2.4 The Sequent Calculus and Pure Explicit Substitutions

In the sequent calculus, the cut rule has always the status of a rule, and this is the only rule that is normally eliminated from any proof to obtain its normal form. It can be given the same interpretation in this setting as in natural deduction, through the typing of an explicit substitution. The difference between interpretation of the sequent calculus and of natural deduction is not in the use of the cut, but in the handling of applications. Since there is no implication elimination rule in LJ, it is impossible to type the standard application in a system based on LJ. This is where pure explicit substitutions come into play: the left implication rule can be used to type the sequentialised, general form of application they use, with the rule:

$$\texttt{let}\ \frac{\Gamma \vdash u : A \qquad \Delta, z : B \vdash t : C}{\Gamma, \Delta, x : A \to B \vdash \texttt{let}\ z = x\ u\ \texttt{in}\ t : C}$$

which simply ensures that the argument u given to the function $x : A \to B$ has the correct type A, and it requires to type t under the assumption that the result z of this application is of type B. The simplest example for a correspondence between functional terms and the sequent calculus is therefore not $\lambda\mathtt{x}$ in this setting, but its variant $\lambda\overline{\mathtt{x}}$. We can now update our picture:

Logic	Computation
proof $\mathscr{P}$ of A in $\mathsf{LJa} \cup \{\mathsf{cut}\}$	closed $\lambda\overline{\mathtt{x}}$-term t of type A
cut on B in $\mathscr{P}$	redex $u[x \leftarrow v]$ in t, with v of type B
cut elimination in $\mathscr{P}$	strong reduction $t \longrightarrow^{*}_{\lambda\overline{\mathtt{x}}} r$

where the sytem $\mathsf{LJa} \cup \{\mathsf{cut}\}$ is an » *additive* « presentation[14] of intuitionistic logic in the sequent calculus. The use of other variants of LJ as the basis for type systems induces a correspondence with some other λ-calculi with pure explicit substitutions which were described previously. In particular, using separate rules for weakening and contraction will have the same effect here than in natural deduction, and leads to consider calculi using an elaborate form of management of resources.

The type system $\mathtt{S}\overline{\mathtt{x}}$ for the $\lambda\overline{\mathtt{x}}$-calculus is given below in Figure 9, and it uses the same rules as $\mathtt{Sx}$, except for the $\mathtt{app}$ which is replaced with $\mathtt{let}$[15] — and the $\mathtt{con}$ rule which will not be used from now on, since we can decide not to use constants, and it is not a particularly interesting rule. This system is syntax-directed and behaves the same as $\mathtt{Sx}$, with a terminating typing process, uniqueness of the computed type, and it provides a matching between proofs and terms — where the sequential application in $\lambda\overline{\mathtt{x}}$ corresponds to the use of the left implication rule.

[14] This system is not completely additive since the succedent of the sequent is not duplicated in the left implication rule, but otherwise it conforms to the policies of duplication of additive systems.

[15] In this variant of the $\mathtt{let}$ rule, the assumption on the type of x is part of the multiset Γ, and this is denoted by $\Gamma \ni x : A \to B$ to avoid explicitly writing this assumption in the premises — where x appears, since it is duplicated when Γ is duplicated.

$$\mathtt{var}\ \frac{}{\Gamma, x:A \vdash x:A} \qquad \mathtt{sub}\ \frac{\Gamma \vdash u:A \quad \Gamma, x:A \vdash t:B}{\Gamma \vdash t[x \leftarrow u]:B}$$

$$\mathtt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \mathtt{let}\ \frac{\Gamma \vdash u:A \qquad \Gamma, z:B \vdash t:C}{\Gamma \ni x:A \to B \vdash \mathtt{let}\ z = x\ u\ \mathtt{in}\ t:C}$$

Figure 9: Type system $\mathrm{S}\overline{\mathrm{x}}$ for the $\lambda\overline{\mathrm{x}}$-calculus

We can now consider the reduction rules of $\lambda\overline{\mathrm{x}}$ and describe how they relate to the rewriting steps on proofs used in the cut elimination procedure for the sequent calculus, as described in Chapter 1. The reduction cases are:

1. The reduction rule $t[x \leftarrow y] \longrightarrow_{\mathtt{ren}} y$ corresponds to the erasure of a cut introducing a lemma reduced to an axiom instance:

$$\begin{array}{l}\mathtt{var}\ \dfrac{}{\Gamma \vdash y:A} \quad \Gamma, x:A \vdash t:B \\ \mathtt{sub}\ \dfrac{}{\Gamma \ni y:A \vdash t[x \leftarrow y]:B}\end{array} \qquad \longrightarrow \qquad \Gamma \ni y:A \vdash t\{y/x\}:B$$

2. The reduction rule $x[x \leftarrow u] \longrightarrow_{\mathtt{var}} u$ corresponds to the replacement of an axiom by the proof of the lemma involved in the cut:

$$\mathtt{sub}\ \frac{\Gamma \vdash u:A \qquad \mathtt{var}\ \dfrac{}{\Gamma, x:A \vdash x:A}}{\Gamma \vdash x[x \leftarrow u]:A} \quad \longrightarrow \quad \Gamma \vdash u:A$$

3. The reduction rule $z[x \leftarrow u] \longrightarrow_{\mathtt{nov}} z$ corresponds to the erasure of the cut and of the proof of the lemma, when an axiom on another formula than the lemma is encountered:

$$\mathtt{sub}\ \frac{\Gamma \vdash u:A \qquad \mathtt{var}\ \dfrac{}{\Gamma, x:A \vdash z:B}}{\Gamma \ni z:B \vdash z[x \leftarrow u]:B} \quad \longrightarrow \quad \mathtt{var}\ \frac{}{\Gamma \ni z:B \vdash z:B}$$

4. The reduction rule $(\lambda y.t)[x \leftarrow u] \longrightarrow_{\mathtt{lam}} \lambda y.t[x \leftarrow u]$ corresponds to the permutation of the cut above an introduction instance, so that the derivation:

$$\mathtt{sub}\ \frac{\Gamma \vdash u:A \qquad \mathtt{lam}\ \dfrac{\Gamma, x:A, y:C \vdash t:D}{\Gamma, x:A \vdash \lambda y.t:C \to D}}{\Gamma \vdash (\lambda y.t)[x \leftarrow u]:C \to D}$$

is turned into the following derivation:

$$\mathtt{sub}\ \frac{\Gamma, y:C \vdash u:A \quad \Gamma, y:C, x:A \vdash t:D}{\mathtt{lam}\ \dfrac{\Gamma, y:C \vdash t[x \leftarrow u]:D}{\Gamma \vdash \lambda y.t[x \leftarrow u]:C \to D}}$$

5. The rule $(\texttt{let}\ y = z\ v\ \texttt{in}\ t)[x \leftarrow u] \longrightarrow_{\texttt{let}} \texttt{let}\ y = z\ v[x \leftarrow u]\ \texttt{in}\ t[x \leftarrow u]$ corresponds to the permutation of the cut above a left implication instance, so that the derivation:

$$\texttt{sub}\ \dfrac{\dfrac{\Gamma \vdash u : A \qquad \texttt{let}\ \dfrac{\Gamma, x : A \vdash v : B \quad \Gamma, x : A, y : C \vdash t : D}{\Gamma, x : A \vdash \texttt{let}\ y = z\ v\ \texttt{in}\ t : D}}{\Gamma \ni z : B \to C \vdash (\texttt{let}\ y = z\ v\ \texttt{in}\ t)[x \leftarrow u] : D}}$$

is turned into the following derivation:

$$\dfrac{\texttt{sub}\ \dfrac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash v : B}{\Gamma \vdash v[x \leftarrow u] : B} \qquad \texttt{sub}\ \dfrac{\Gamma, y : C \vdash u : A \quad \Gamma, y : C, x : A \vdash t : D}{\Gamma, y : C \vdash t[x \leftarrow u] : D}}{\texttt{let}\ \dfrac{}{\Gamma \ni z : B \to C \vdash \texttt{let}\ y = z\ v[x \leftarrow u]\ \texttt{in}\ t[x \leftarrow u] : D}}$$

and the complete typing derivation above the premise $\Gamma \vdash u : A$ needs to be duplicated, and plugged above both copies of this premise.

6. The rule $(\texttt{let}\ y = x\ v\ \texttt{in}\ t)[x \leftarrow \lambda z.u] \longrightarrow_{\texttt{apd}} t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u]$ corresponds to the permutation of the cut above a left implication instance where the lemma is the formula decomposed, so that the derivation:

$$\dfrac{\texttt{lam}\ \dfrac{\Gamma, z : A \vdash u : B}{\Gamma \vdash \lambda z.u : A \to B} \qquad \texttt{let}\ \dfrac{\Gamma, x : A \to B \vdash v : A \quad \Gamma, x : A \to B, y : B \vdash t : C}{\Gamma, x : A \to B \vdash \texttt{let}\ y = x\ v\ \texttt{in}\ t : C}}{\texttt{sub}\ \dfrac{}{\Gamma \vdash (\texttt{let}\ y = x\ v\ \texttt{in}\ t)[x \leftarrow \lambda z.u] : C}}$$

is turned into the following derivation:

$$\dfrac{\texttt{lam}\ \dfrac{\Gamma, z : A \vdash u : B}{\Gamma \vdash \lambda z.u : A \to B} \qquad \overset{\mathscr{D}_1}{\nabla} \quad \Gamma, x : A \to B \vdash t[y \leftarrow u[z \leftarrow v]] : C}{\texttt{sub}\ \dfrac{}{\Gamma \vdash t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u] : C}}$$

where $\mathscr{D}_1$ is the derivation:

$$\texttt{sub}\ \dfrac{\dfrac{\Gamma, x : A \to B \vdash v : A \quad \Gamma, x : A \to B, z : A \vdash u : B}{\Gamma, x : A \to B \vdash u[z \leftarrow v] : A} \qquad \Gamma, x : A \to B, y : B \vdash t : C}{\Gamma, x : A \to B \vdash t[y \leftarrow u[z \leftarrow v]] : C}$$

7. The reduction rule $t[x \leftarrow \texttt{let}\ y = z\ v\ \texttt{in}\ u] \longrightarrow_{\texttt{out}} \texttt{let}\ y = z\ v\ \texttt{in}\ t[x \leftarrow u]$ corresponds to the permutation of the cut above a left implication instance located in the proof of the lemma, so that the derivation:

$$\texttt{let}\ \dfrac{\texttt{sub}\ \dfrac{\Gamma \vdash v : B \quad \Gamma, y : C \vdash u : A}{\Gamma \vdash \texttt{let}\ y = z\ v\ \texttt{in}\ u : A} \quad \Gamma, x : A \vdash t : D}{\Gamma \ni z : B \to C \vdash t[x \leftarrow \texttt{let}\ y = z\ v\ \texttt{in}\ u] : D}$$

$$\mathtt{var}\,\frac{}{\Gamma, x:A \vdash x:A} \qquad \mathtt{sub}\,\frac{\Gamma \vdash u:A \quad \Gamma, x:A \vdash t:B}{\Gamma \vdash t[x \leftarrow u]:B}$$

$$\mathtt{rem}\,\frac{\Gamma \vdash t:B}{\Gamma, x:A \vdash t:B} \qquad \mathtt{dup}\,\frac{\Gamma, x:A, y:A \vdash t_{[y/x]}:B}{\Gamma, x:A \vdash t:B}$$

$$\mathtt{lam}\,\frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \mathtt{let}\,\frac{\Gamma \vdash u:A \qquad \Gamma, z:B \vdash t:C}{\Gamma \ni x:A \to B \vdash \mathtt{let}\ z = x\ u\ \mathtt{in}\ t:C}$$

Figure 10: Type system $\mathtt{S}\overline{\mathtt{s}}$ for the $\lambda\overline{\mathtt{s}}$-calculus

is turned into the following derivation:

$$\mathtt{let}\,\frac{\Gamma \vdash v:B \qquad \mathtt{sub}\,\dfrac{\Gamma, y:C \vdash u:A \quad \Gamma, y:C, x:A \vdash t:D}{\Gamma, y:C \vdash t[x \leftarrow u]:D}}{\Gamma \ni z:B \to C \vdash \mathtt{let}\ y = z\ v\ \mathtt{in}\ t[x \leftarrow u]:D}$$

Notice that in the $\lambda\overline{\mathtt{x}}$-calculus, the last case, where a left implication instance is moved down from the left premise of a cut, under this cut, is not always accepted as a valid rewriting, since the `out` rule can be applied only when the term t under the substitution $[x \leftarrow u]$ is itself an application of x to some other term v. This is necessary to preserve the confluence, and this can also be seen on the logical level, since an unrestricted permutation of rules from the left premise of a cut would lead to different proofs as results of the cut elimination procedure.

Refined calculi. As in the setting of standard calculi with explicit substitution, based on natural deduction, the basic $\lambda\overline{\mathtt{x}}$-calculus can be refined to treat resources more carefully, and type systems improving on $\mathtt{S}\overline{\mathtt{x}}$ can be defined for these calculi. In particular, the type system $\mathtt{S}\overline{\mathtt{s}}$, shown above in Figure 10, allows to establish a correspondence between the multiplicative presentation of LJ and the $\lambda\overline{\mathtt{s}}$-calculus with pure explicit substitutions. The system $\mathtt{S}\overline{\mathtt{s}}$, just as `Ss`, is not syntax-directed because the calculus allows free duplication of substitutions, so that the structural rules can be applied at any point.

The reduction rules of $\lambda\overline{\mathtt{s}}$ correspond to the same cases as shown above for $\lambda\overline{\mathtt{x}}$, but where the typing assumptions are treated in a multiplicative way. The two new rules `rem` and `dup` interact with the rule `sub` exactly as in the $\lambda\mathtt{s}$-calculus:

$$\mathtt{sub}\,\frac{\Gamma \vdash u:A \qquad \mathtt{rem}\,\dfrac{\Delta \vdash t:B}{\Delta, x:A \vdash t:B}}{\Gamma, \Delta \vdash t[x \leftarrow u]:B} \qquad \longrightarrow \qquad \mathtt{rem}^*\,\frac{\Delta \vdash t:B}{\Gamma, \Delta \vdash t:B}$$

and contraction is also equivalent to the permutation and duplication of a cut above a contraction, when this contraction affects the cut formula.

3 Linear Logic and Resources

Historically, the development of structural proof theory, in the frameworks defined by Gentzen, Prawitz and others, was tied to the development of logical approaches to computation, mainly through the Curry-Howard correspondence and its various extensions. In particular, the treatment of erasure and duplication of assumptions, its relation to intuitionism and constructivism, and its use in the design of various functional interpretations can be seen as a motivation for the introduction of *linear logic* [Gir87], where this question is made explicit by the strict distinction between connectives allowing erasure and duplication, and linear connectives. As it implies that one can deal explicitly with the availability of » *enough* « copies of a formula, for example, linear logic is often described as the logic of *resources*, and motivated many investigations around this topic, in terms of different logical approaches to computation, such as logic programming [HM94] and typed functional languages with complexity bounds [Gir98, Laf04]. The general methodology used in linear logic, proceeding by decomposition of existing system, through a careful structural analysis, and validating this by the recomposition of the original system, has been influential, for example in the development of the deep inference methodology.

3.1 A Linear Decomposition of Classical Logic

The basic idea of disallowing erasure and duplication of usual formulas, and then allowing that only when a particular operator is used, comes from the intuitionistic setting, through the analysis of models of System F. Indeed, the usual interpretation of the type $A \to B$ is a function taking an argument A and returning some result B, disregarding how many copies of the argument are needed to actually compute the result. The decomposition performed in linear logic is written $!A \multimap B$, and this can be interpreted as a function asking for » *as many copies of A as required* « and returning exactly one copy of the resulting B.

However, the most general, and original presentation of linear logic is classical, using disjunctive and conjunctive connectives rather than the linear implication $\multimap$, and it can be described as a decomposition of the standard presentation of classical logic in the sequent calculus.

Multiplicatives and Additives. The linear decomposition of LK is based on the observation, described in Chapter 1, that there are two alternative presentations, where weakening and contraction are handled differently. For example, $\wedge$ could be described by an inference rule with built-in contraction, or by a rule that requires to split the context, so that contraction would be implemented in another inference rule. This can be expressed as follows:

$$\wedge\,\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B} \qquad \equiv \qquad \wedge\,\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \qquad + \qquad \mathsf{cont}\,\frac{\vdash \Gamma, A, A}{\vdash \Gamma, A}$$

and the idea is to allow the use of both kinds of rules in the same system, through a distinction between a multiplicative $\wedge$ and an additive $\wedge$.

In linear logic, each classical connective and unit is divided into its two possible variants, multiplicative or additive. Syntactically, this doubles the number of logical symbols, and allows to control the use of variants of the standard classical rules.

Classical	Mutiplicative	Additive
$\vee$	$\parr$	$\oplus$
$\wedge$	$\otimes$	$\&$
$\top$	1	$\top$
$\bot$	$\bot$	0

The connectives of linear logic[16] also inherit the nice symmetries of conjunction and disjunction in classical logic, within their structural category, so that $\parr$ is the dual of $\otimes$ while $\oplus$ is the dual of $\&$, and for the units we have 1 dual to $\bot$ while $\top$ is dual to 0. However, this decomposition and the rules corresponding to the two structural presentations, even used together in the same system, does not allow to define a system that would be complete with respect to classical logic [Hug10].

Exponentials and Unbounded Behaviour. The missing piece in the definition of a complete decomposition of classical logic is the ability to erase or to duplicate formulas, as allowed by the weakening and contraction rules in LK. In linear logic, the formulas that can be erased or duplicated are marked using a unary connective, so that weakening and contraction can be defined as rules which apply only on the formulas of the shape $?A$ — and to preserve symmetry, the dual of ? is also defined and denoted by !¹⁷ so that we can write $(?A)^{\bot} = !A^{\bot}$.

These two new connectives are called the *exponentials*, and they have particular properties with respect to the multiplicative and additive connectives. Indeed, they represent weakening and contraction, and as mentioned these structural operations are exactly the difference between the two basic categories of linear logic formulas, as expressed in the following equations:

$$!(A \& B) \equiv !A \otimes !B \qquad \text{and} \qquad ?(A \oplus B) \equiv ?A \parr ?B$$

which are valid equivalences in linear logic. This makes explicit the fact that the exponentials are needed to relate multiplicative and additive connectives, in order to have a system complete for classical logic — that is, a system which can simulate the LK sequent calculus. Indeed, using a mixture of the multiplicative and additive presentations of conjunction and disjunction, it is impossible to define a classically complete system [Hug10], because there are formulas in which it is necessary to consider some connectives as a blend of multiplicative and additive flavours.

[16] The symbols used for connectives in linear logic are meant to express their interpretations, and have become standard in the literature: $\parr$ is called » *par* « and $\otimes$ is called » *tensor* « while $\oplus$ is called » *plus* « and $\&$ is called » *with* « and the units are simply read » *one, bottom, top* « and » *zero* «.

[17] In the linear logic literature, the symbol ? is called » *why not* « and the symbol ! is called » *of course* «, or sometimes » *bang* « — this raises the question of a name for ? that would be short: some say » *plic* «.

$$\mathsf{ax}\ \frac{}{\vdash A, A^\perp} \qquad 1\ \frac{}{\vdash 1} \qquad \parr\ \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$$

$$\mathsf{cut}\ \frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta} \qquad \bot\ \frac{\vdash \Gamma}{\vdash \Gamma, \bot} \qquad \otimes\ \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}$$

Figure 11: Inference rules for system $\mathsf{MLL} \cup \{\mathsf{cut}\}$

3.2 Fragments of Linear Logic

The design of linear logic has the nice property that its sequent calculus LL can be divided into fragments, with different purposes, that can be composed in different ways or used all together as the calculus for full linear logic.

Purely multiplicative fragment. The first and most foundamental fragment of linear logic is called MLL, and it deals only with multiplicative connectives. This is the heart of linear logic, since MLL contains both the identity axiom, and the cut rule when it is used. The inference rule for this sequent calculus are given above in Figure 11, and it can be observed immediately that through the translation of linear formulas into classical formulas, this is nothing else than LK without weakening and contraction.

Example 3.1. *Below are shown two proofs in* MLL *illustrating the use of inference rules, in particular for units in the proof on the left, and the interaction between dual connectives in the proof on the right.*

$$\otimes\ \frac{1\ \frac{}{\vdash 1} \qquad \parr\ \frac{\bot\ \frac{\mathsf{ax}\ \frac{}{\vdash A, A^\perp}}{\vdash A, A^\perp, \bot}}{\vdash A, A^\perp \parr \bot}}{\vdash A, 1 \otimes (A^\perp \parr \bot)} \qquad \parr\ \frac{\otimes\ \frac{\mathsf{ax}\ \frac{}{\vdash B, B^\perp} \quad \mathsf{ax}\ \frac{}{\vdash A^\perp, A}}{\vdash A^\perp, B, B^\perp \otimes A}}{\vdash A^\perp \parr B, B^\perp \otimes A}$$

It is important to notice the consequences of linearity, as shown in the two derivations below — on the left, the $B^\perp$ *formula prevents the use of an axiom on A, and in the one on the right, the use of the cut illustrates how the meta-level is also made linear, by splitting contexts in a multiplicative way rather than duplicating formulas.*

$$\otimes\ \frac{\mathsf{ax}\ \frac{}{\vdash B^\perp, B} \qquad \parr\ \frac{\vdash A, B^\perp, A^\perp}{\vdash A \parr B^\perp, A^\perp}}{\vdash A \parr B^\perp, B^\perp, B \otimes A^\perp} \qquad \mathsf{cut}\ \frac{\mathsf{ax}\ \frac{}{\vdash A \otimes B, A^\perp \parr B^\perp} \qquad \otimes\ \frac{\mathsf{ax}\ \frac{}{\vdash A, A^\perp} \quad \vdash B}{\vdash A \otimes B, A^\perp}}{\vdash A \otimes B, A^\perp}$$

$$\top\ \frac{}{\vdash \Gamma, \top} \qquad \oplus_{\mathsf{L}}\ \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B} \qquad \oplus_{\mathsf{R}}\ \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B} \qquad \&\ \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}$$

Figure 12: Inference rules of the additive fragment of LL

Multiplicative, additive fragment. Beyond the core multiplicative fragment of the logic, the first step is to use additive connectives. However, there is no purely additive fragment, because there is no equivalent of the identity rule there. In order to use the rules for additives, we add them to the multiplicative fragment to define the MALL system. The inference rules for the additive connectives extending MLL into this new calculus are shown above in Figure 12.

In this fragment, the $\top$ unit is the source of problems, because its associated inference rule erases the other formulas in a sequent containing it, although these formulas are separated by commas — and the comma is logically equivalent to the $\parr$ connective, which is multiplicative. In particular, the handling of permutations involving $\top$ is complicated, since permuting this rule downwards in a proof erases parts of the proof, and this is an irreversible change.

Example 3.2. *Below are shown two proofs in the* MALL *fragment, illustrating how the additive connectives behave and interact with each other. The one on the left shows how the additive truth unit allows to close branches without applying the identity rule, and on the right we can see that an additive disjunction must match an additive conjunction — if the $\oplus$ was a $\parr$ there, the proof could not be completed.*

$$\dfrac{\dfrac{\top\ \dfrac{}{\vdash B, A, \top}}{\parr\ \dfrac{}{\vdash B, A \parr \top}}}{\oplus_{\mathsf{L}}\ \dfrac{}{\vdash B, (A \parr \top) \oplus 0}} \qquad \dfrac{\dfrac{\mathsf{ax}\ \dfrac{}{\vdash A, A^{\perp}}}{\oplus_{\mathsf{L}}\ \dfrac{}{\vdash A \oplus B, A^{\perp}}} \quad \dfrac{\mathsf{ax}\ \dfrac{}{\vdash B^{\perp}, B}}{\oplus_{\mathsf{R}}\ \dfrac{}{\vdash A \oplus B^{\perp}, B}}}{\&\ \dfrac{}{\vdash A \oplus B^{\perp}, A^{\perp} \,\&\, B}}$$

Then, the two derivations below cannot be completed because of linearity. On the left, the sequent $\vdash 0, B$ has no proof because there is no rule for 0 — it cannot be removed by weakening. On the right, the derivation shows that even with additive formulas, the cut is multiplicative, so that the context is linearly splitted.

$$\dfrac{\dfrac{\mathsf{ax}\ \dfrac{}{\vdash A, A^{\perp}}}{\oplus_{\mathsf{L}}\ \dfrac{}{\vdash A \oplus 0, A^{\perp}}} \quad \dfrac{\vdash 0, B}{\oplus_{\mathsf{R}}\ \dfrac{}{\vdash A \oplus 0, B}}}{\&\ \dfrac{}{\vdash A \oplus 0, A^{\perp} \,\&\, B}} \qquad \dfrac{\dfrac{\mathsf{ax}\ \dfrac{}{\vdash A, A^{\perp}}}{\oplus_{\mathsf{L}}\ \dfrac{}{\vdash A, A^{\perp} \oplus B^{\perp}}} \quad \dfrac{\vdash B^{\perp}, A^{\perp} \quad \mathsf{ax}\ \dfrac{}{\vdash B^{\perp}, B}}{\&\ \dfrac{}{\vdash B^{\perp}, A \,\&\, B}}}{\mathsf{cut}\ \dfrac{}{\vdash A, B^{\perp}}}$$

Notice that the derivation on the right could not be modified in a way that would allows to close it, since that would require to duplicate the formula A in the conclusion, and to erase the $B^{\perp}$ in the left branch of the $\&$ rule instance.

$$\mathsf{we}\,\frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad \mathsf{de}\,\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \qquad \mathsf{ce}\,\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \qquad \mathsf{pe}\,\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}$$

Figure 13: Inference rules of the exponential fragment of LL

Exponentials and Full Linear Logic. The last set of rules to add to the system is the one dealing with exponential connectives, and as before, this is not a proper fragment that can be used alone, but rather an extension that can be inserted into either MLL or MALL. The set of inference rules used for exponential connectives are shown above in Figure 13, and they can be used in the definition of two distinct systems. When added to MALL, these rules complete it into the LL sequent calculus for full linear logic, which provides the complete linear decomposition of classical logic. When added to MLL, these rules yield the MELL system, where weakening and contraction are available for formulas marked with exponentials, but where an erasure or a duplication can never happen as a byproduct of the treatment of some connective — as it happens in additive rules.

The exponential fragment is more complex than the others. In particular, the *promotion* rule pe is unusual for the sequent calculus, because it is *» non-local «* in the sense that it can only be applied if all formulas in the sequent except $!A$ are of the shape $?B$ — this is expressed by writing $?\Gamma$ in the rule. Moreover, from a proof construction viewpoint, the other three rules can seem problematic because they have the same conclusion, so that there is a choice to be done concerning the formula $?A$, to either erase it, linearise it or duplicate it.

Example 3.3. *Below are shown two proofs in* LL *illustrating the use of the rules for exponentials, showing how the modalities interact, and how the different possibilities of applying rules can be used.*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\mathsf{ax}}{\vdash A, A^\perp, ?A^\perp}\,\mathsf{we}}{\vdash A, ?A^\perp, ?A^\perp}\,\mathsf{de}}{\vdash A, ?A^\perp}\,\mathsf{ce} \qquad \dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\mathsf{ax}}{\vdash ?A, A^\perp}\,\mathsf{de}}{\vdash ?A, !A^\perp}\,\mathsf{pe}$$

Then, the two derivations below provide examples of how the exponentials correspond to a bridge between the multiplicative and additive fragments, allowing here to use the additive disjunction and conjunction as their multiplicative counterparts.

$$\dfrac{\dfrac{\dfrac{\vdash A, B}{\vdash A, ?(A \oplus B)}\,\mathsf{de};\oplus_\mathsf{R}}{\vdash ?(A \oplus B), ?(A \oplus B)}\,\mathsf{de};\oplus_\mathsf{L}}{\vdash ?(A \oplus B)}\,\mathsf{ce} \qquad \dfrac{\dfrac{\dfrac{\vdash A, C}{\vdash A, ?B, C}\,\mathsf{we}}{\vdash ?A, ?B, C}\,\mathsf{de} \quad \dfrac{\dfrac{\vdash B, D}{\vdash ?B, D}\,\mathsf{de}}{\vdash ?A, ?B, D}\,\mathsf{we}}{\vdash ?A, ?B, C \,\&\, D}\,\&$$

3.3 Computational Significance

Linear logic is relevant to the logical approach of computation in several ways. Its definition was originally motivated by the fine-grained analysis of the semantics of the λ-calculus which lead to the introduction of coherence spaces [GLT89], and it is often described as a refinement of intuitionistic logic although it is » *classical* « in the sense that it is highly symmetric and has an involutive negation. In particular, the cut elimination procedure of the LL sequent calculus is confluent, whereas in LK it is not confluent, as shown by Lafont's counter-example — where a weakening is applied on both occurrences of a cut formula. In general, the improvement of linearity over the traditional intuitionistic and classical logics lies in the expressivity of the formulas, which describe in more details the behaviour of the corresponding inference rules, in particular when it comes to the handling of *resources*.

Proofs as programs. As a constructive logic, in the tradition of intuitionism, linear logic was a good candidate to be the basis of a correspondence following the *Curry-Howard* tradition, where linear proofs would be interpreted as programs in some term calculus and cut elimination in LL would represent computation in this calculus. Along these lines, there are different possible approaches, among which the interpretation of the intuitionistic fragment of linear logic — where disjunction in its multiplicative form $\parr$ is replaced with the linear implication $\multimap$ and sequents are restricted accordingly — as a type system for variant of the λ-calculus, and the interpretation of classical LL as type system for some process algebra, were studied intensively in the hope of establishing linear logic as a foundational tool in the field of computational logic [Abr93, BS94].

Beyond the distinction between multiplicative and additive connectives, which allows intersting distinctions in the constructs of a language based on linear logic, the exponentials are crucial in the definition of a » *linear* « variant of the λ-calculus, where linearity is understood as a control of erasure and duplication rather than a simple ban, which would induce a very weak calculus. The most important rule of LL in this regard is promotion, since it enforces a restriction on the variables that can be used in the typing context of a term typed with a formula $!A$. Intuitively, and through the observation that a cut associates formulas of the shape $!B$ and $?B$, this means that only variables attached to a *duplicable* term can be used in a duplicable term — since $!A$ is the type of a term that can be used several times.

Proof search. The sequent calculus LL for linear logic is also of great interest in terms of proof construction, and has therefore received a lot of attention in the studies where proof search is seen as computation [HM94, BG96]. Indeed, in linear logic, the distinction between the multiplicative and the additive treatments of connectives is built inside the formulas, so that the expressivity is improved by allowing to mix these two styles. Moreover, the shape of the LL sequent calculus has lead to the introduction of *focused* proofs [And92], a refinement of the notion of *uniform* proofs [MNPS91] which has taken a very important role in the structural approach to logic programming. The focusing technique has revealed to be a deep result in proof-theory, was extended to various other logics, and impacted both the logical approach to computation and functional interpretations.

4 Proof Search as Logical Computation

The purely logical approach to computation called *logic programming* is based on the idea that programs can be encoded in the formulas of a logic, so that proving this formula is equivalent to the computation of the program. This is the basis of the Prolog language [MW88], and that can be supported on the theoretical level by several approaches. The most common description of logic programming defines it as an application of the *resolution* method [Rob65], which is indeed the basis for most implementation of Prolog, but this viewpoint has drawbacks. In particular, it makes the extension of the language difficult, if one wants to retain the purity of the logical interpretation.

Logic programming can also be described in the terms of structural proof theory [BG03], and this approach has been mostly based in the framework of the sequent calculus. A benefit of this approach is that various logics can be used, possibly with different languages of formulas that can provide for the right level of expressivity, depending on the task. Following this methodology, the proof construction process of a sequent calculus is seen as computation, but usually, not all possible proofs of a system are considered as valid computation trace. In order to give a sensible operational meaning to to a program written as a formula, only particular rules and instances should be accepted in a system. This has lead to the definition of the notion of *uniform proofs* [MNPS91], an important milestone in the proof theoretical presentation of logic programming.

4.1 Computational Interpretations of Formulas

The basic layer of logic programming is the syntax for formulas, which is also the syntax for programs. Intuitively, the construction of a proof for some formula in a sequent calculus system does not produce any result, or any output, but simply says whether this formula is provable or not. However, in a system using quantifiers, there is one important result produced by this process: the instantiations chosen for variables when an existential formula $\exists x.A$ is proved, which are the *witnesses* of the provability of this formula — an instantiation is also produced when treating a universal quantification, when it is used as an hypothesis. Therefore, the program *» produce an object x satisfying the property P «* can be written *» is there an object x such that the property $P(x)$ holds ? «* and such queries are the basic way of obtaining the result of a computation in logic programming.

A logic program is a set of *clauses*, which are logical formulas of some particular shape. The most common, intuititive kind of clause is:

$$\forall x_1.\cdots\forall x_n.(b_1 \wedge \cdots \wedge b_k \rightarrow a)$$

which represents the piece of program stating that under any instantiation of the variables x_1 to x_n, the fact a holds if all of the facts b_1 to b_k hold as well. Then, such clauses are gathered in the antecedent of a sequent to form a complete logic program P, and the succedent of the sequent is the query that we try to answer under the program P.

The execution of the query under the given program is simply the search for a proof of this sequent, and inference rules in this setting are nothing more than the transition of some *logical machine*, which implements this logical language the same way as an abstract machine can implement a λ-calculus. Once the proof has been built, the substitution created by applying rules for quantifiers contains the expected result — if a proof was found, because the search could fail, and failure does not always mean that there is no result, but rather that it could not be found by the logical engine.

In this setting, any refinement of the standard intuitionistic and classical logics provides improvements in the language and the operational behaviour of programs. For example, the use of linear logic allows to control the way resources can be used in clauses, so that one can write $(b \otimes b) \multimap a$ to specify that the resource b is needed twice to produce the fact a. The rich language of linear logic [HM94] provides a gain in expressivity and allows to treat problems that could not be solved otherwise, and other extensions such as the ∇ quantifier offer solutions for complex problems — involving binders, for example [TM04].

4.2 Normal Forms and Proof Search

The standard proof systems in the sequent calculus, such as LJ, LK or LL, are not used directly in proof search, but modified to avoid potential problems such as the possibility to apply freely structural rules. The notion of uniform proofs [MNPS91] is based on the study of the deductive constructs providing a reasonable framework for executing logic programs, and this lead to the notion of *focusing* [And92], which defines a particularly well-behaved normal form for linear logic proofs, and allows for structured and efficient proof search. To illustrate this, we can consider the case of uniformity in the intuitionistic setting.

Definition 4.1. *In an intuitionistic sequent calculus, a proof $\mathscr{P}$ is* uniform *if and only if for any sequent $\Gamma \vdash A$ in $\mathscr{P}$, if A is not an atomic formula then this sequent is the conclusion of a right rule.*

The focusing methodology is more general, and consists in the definition of a focused variant of a given sequent calculus, where annotations are introduced to restrict the application of inference rules, based on the permutability properties of these rules and the notion of *polarity* of connectives. According to this idea, there are two categories of connectives in linear logic:

- The *negative* connectives $\parr$, $\with$ and ?, and the units $\bot$ and $\top$, for which the inference rules are invertible, and can thus be applied eagerly.
- The *positive* connectives $\otimes$, $\oplus$ and !, and both units 1 and 0, for which the inference rules are not invertible, and should be applied only after other rules have been used to prepare the context.

An intuitive idea on negative and positive connectives is that negatives are » *easy* « and require no intelligence, while positives need a » *clever guess* « to be decomposed the right way, but the situation is actually slightly more complicated than this.

More precisely, it is preferable to consider the permutability of inference rules rather than the polarity of connectives. We can make the distinction between two categories of rules, related to the notion of negative and positive but more specific to the focusing approach:

- The *asynchronous* rules have *full permutability*, so they can be permuted with any other rule instance, provided that the formula they decompose is present in the sequent after permutation.
- The *synchronous* rules have *weak permutability*, they can only be permuted with other synchronous rule instances.

The standard syntax of focused systems follows Andreoli [And92], introducing two arrows $\Downarrow$ and $\Uparrow$ to signal the synchronous or asynchronous state of a sequent, respectively, and it uses a *stoup* to store formulas that can be duplicated or erased because they have be extracted from an exponential ? modality. There are then two kinds of *triadic* sequents:

$$\vdash \Gamma \mid \Delta \Downarrow A \qquad \text{and} \qquad \vdash \Gamma \mid \Delta \Uparrow \Psi$$

where Γ is the multiset of duplicable formulas, in the stoup on the left of $\mid$ while Δ is a multiset of linear formulas. On the right of the arrow, there is exactly one formula if the the sequent is in a synchronous state denoted by $\Downarrow$ and a multiset of formulas if the sequent is in an asynchronous state, denoted by the $\Uparrow$ arrow. Then, some rules will depend on the nature of a formula in the sequent, so that we need to use the two categories described below:

$$\begin{aligned} P, Q &::= a \mid 1 \mid A \otimes B \mid 0 \mid A \oplus B \mid !A \\ N, M &::= \overline{a} \mid \bot \mid A \parr B \mid \top \mid A \& B \mid ?A \end{aligned}$$

and we also write P° to denote a formula which is either a positive P or a negative atom. The triadic focused system LLF is defined by the set of rules shown below in Figure 14. It is separated in three fragments: the decision and reaction rules are required to handle the state of sequents and start or end *phases*, which are either synchronous or asynchronous and are performed by the rules of the corresponding fragments.

Remark 4.2. *The rules of* LLF *are such that negative formulas are treated only in the asynchronous rules and positive formulas are treated in the synchronous rules, but it is not directly similar to the basic* LL *system, since it uses triadic sequents where the modality* ? *can be immediately treated only because the stoup allows the duplication of formulas. This » trick « reflects to the fact that exponentials do not fit perfectly the scheme of negative and positive formulas, and in particular that the treatment of* ? *is complex, from the viewpoint of proof search.*

The idea behind this focused system was originally motivated by proof search considerations. It defines a particular strategy to build a proof, where all negative connectives are first maximally decomposed, and then one positive is picked and maximally decomposed. Some new negative formulas might have been added to the sequent by this decomposition, and proof search goes on as it started, and this is repeated until the proof is complete.

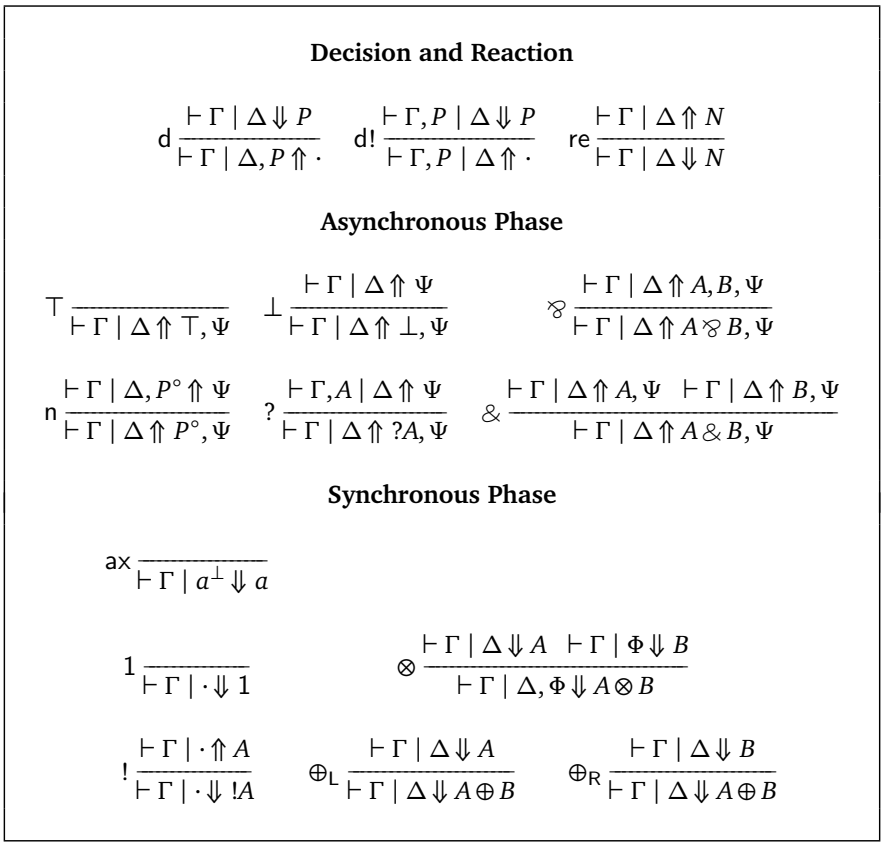


Figure 14: Inference rules for the triadic system LLF

The strategy enforced by the focused sequent calculus corresponds to a kind of cyclic decomposition of LL proofs, where all the complex operations of search are concentrated in the synchronous phases, while asynchronous phases correspond to eager decomposition of connectives, which can be performed because the rules of this fragment are all invertible. In this scheme, the stoup is used to delay the choice of treatment for formulas of the shape $?A$, which are automatically duplicated or erased when needed, in other rules, since this part of a sequent is treated additively in branching rules as in the axiom rules.

Remark 4.3. *Here, we have defined the atoms as basically positive, and negated atoms are negative formulas. However, one can choose freely the polarity of all atoms in a formula, as long as this* bias assignment *is consistent with negation, so that a and $\overline{a}$ have opposite polarities [MS07].*

The focused normal form of proofs in LL, described by this LLF restriction, is quite strong in the sense that many proofs are made invalid — they correspond to the proofs that can be forgotten during proof search, because their structure is not well-organised and they cannot be obtained by following the strategy described above. The somewhat surprising, and important result, is the completeness of this normal form, called the *focusing* result for LL.

Proposition 4.4. *Given a formula A of linear logic, if there is a proof of the sequent $\vdash A$ in* LL*, then there is a proof of $\vdash \cdot \mid \cdot \Uparrow A$ in the focused* LLF *system.*

There are several different proofs of this result, using different approaches. The original proof given by Andreoli [And92] is directly based on permutations of rule instances, and is quite tedious. Other proofs include the one given based on the cut elimination result [Lau04] and a modular proof based on graphs [MS07]. In all of these proofs, the permutability properties of the inference rules of LL are crucial to the result, since they reflect the behaviour of linear connectives and justifies the restrictions imposed by focusing.

The focusing technique has been extended to other logics, starting with both of the standard intuitionistic and classical sequent calculi LJ [LM07] and LK [LM09]. Notice that all these results are closely related to the studies of polarities in linear, intuitionistic and classical logic [Lau02], and some systems using polarities have the same underlying ideas than focused systems [Gir91].

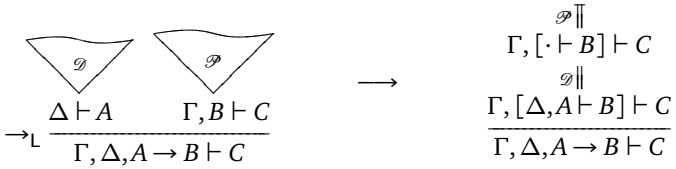
PART 2

Intuitionistic Logic in Deep Inference

Chapter 3

Intuitionistic Logic in Nested Sequents

In this chapter, we introduce a family of intuitionistic proof systems following the principles of nested deduction, in the setting of the nested sequents formalism where the deep inference methodology is tamed by dividing the representation of proofs into the object logical level, and the deductive meta-level. Syntactically, this means that we present a generalisation of the sequent calculus LJ where sequents can appear nested inside other sequents, but where most inference rules are similar to the standard ones. The conceptual difference with a » *shallow* « calculus is simply that *branching* is replaced with *nesting*, as illustrated by the figure below.



In order to illustrate the many possibilities offered in a nested deduction setting regarding the design of a set of inference rules, we present various systems where structural rules are either built inside other rules or used separately. Moreover, we show how these systems relate to the usual sequent calculus LJ, and prove that they are sound and complete with respect to intuitionistic logic.

Then, we study the most foundamental property of intuitionistic systems, from the perspective of structural proof theory: the ability to transform a proof that does not respect the subformula property into a normal proof where every rule instance is *analytic*. In the sequent calculus and in the basic nested sequent presentation, this is cut elimination and it consists in our new systems in a series of permutations of a cut instance upwards in a proof, until it disappears. However, we also present an extension of these systems incorporating the principles of symmetry found in deep inference, where all the rules have a dual. This yields a proof transformation that is completely local — this is not the case in the basic cut elimination for nested sequents —, and prove that it allows to remove from any proof in the symmetric system all the rules dual to the basic rules.

1 Intuitionistic Nested Sequent Systems

We present here a family of proof systems for the purely implicative fragment of intuitionistic logic which are based on the traditional sequent calculus LJ [Gen34], but incorporate the *deep inference* methodology [Gug07] in the sense that sequents can be nested [Brü10] one inside another. Unlike most systems presented in a deep inference setting, these ones are not presented as symmetric systems [Brü03] with two fragments, called *up* and *down*, dual to each other such that the up fragment is admissible for the down fragment. Instead, only the cut, dual to the identity rule, is provided and can be shown admissible *via* the cut elimination procedure — or simply by translation with a cut-free system. Finally, these systems are closer to the *calculus of structures* than usual nested sequents systems in the sense that they use nesting in place of branching, and thus all inference rules have only one premise, and derivations are sequences of inference rule instances.

1.1 Basic Definitions

All the systems we will present here are variations of the basic JN system, and share the same structure and basic definitions. First we need a countable set of *atoms*, denoted by small latin letters such as a, b, c, and the connective $\to$ for intuitionistic implication. We will not use any unit, nor quantifiers, so that the arrow is the only connective we need for implicative intuitionistic logic at the propositional level.

Then, the sequents we will use are an extension of usual intuitionistic sequents where nesting is allowed, in the sense that a sequent can appear on the left-hand side. Thus, a sequent is a pair of an antecedent and a formula, where an antecedent is a finite, possibly empty multiset of sequents, separated by comma.

Definition 1.1. *The* formulas *of intuitionistic logic,* nested sequents *of our systems and* sequent antecedents *are defined by the following grammar:*

$$A,B \ ::= \ a \mid A \to B \qquad \delta \ ::= \ \Gamma \vdash A \qquad \Gamma \ ::= \ \delta_1, \cdots, \delta_n$$

We use capital greek letters such as Γ, Δ, Ψ to denote antecedents and small greek letters such as δ, κ, υ for nested sequents. On the notational level, we use $[\cdot]$ as parentheses around nested sequents, and the short notation A for a nested sequent $\vdash A$ where the antecedent is empty, so that for example:

$$\Gamma, [\Delta, [\vdash A] \vdash B], [\vdash C] \vdash D \qquad \text{is written as} \qquad \Gamma, [\Delta, A \vdash B], C \vdash D$$

Being in a » *deep inference* « setting means having the ability to apply inference rules inside a context — that is, within another sequent, which could be itself on the left-hand side of some other sequent, and so on. However, to simplify notations and make it clear where inference rules can be applied, we consider only *positive* contexts here, which are those located on the left-hand side of an even number of nested sequents.

Definition 1.2. *The* contexts *of our systems are nested sequents with a hole* $\{\}$ *meant to be filled by another nested sequent, and are defined by the following grammar:*

$$\xi \ ::= \ \{\} \mid \Gamma, [\Delta, \xi \vdash A] \vdash B$$

Remark 1.3. *The important property of contexts is that they preserve polarity, so that for example, a context plugged in a negative position has a hole in negative position.*

Contexts will be denoted as $\xi\{\ \}$ or $\zeta\{\ \}$, so that for example $\xi\{\delta\}$ is the context ξ where the hole has been replaced by the nested sequent δ. We can also extend the definition of contexts to several holes, and such a context ξ is denoted by $\xi\{\ \}^+$. Moreover, we can specify a family of elements located in all the holes of a context using indices such as i, j, k, and a notation similar to the one for sums:

$$\xi_i\{\delta_i\}^+ \;=\; \xi\{\delta_1\}\cdots\{\delta_n\} \qquad \text{for some } n \in \mathbb{N} \tag{5}$$

and we will usually not write the annotation on the context, if there is no ambiguity on the indices. Then, we can define inference rule instances as an instantiation of a rule inside some context, obtained by instantiating the schematic variables in its premise and conclusion in the hole of this context, as done in Chapter 1.

The intuitionistic proof systems that we will define in this chapter enjoy a less complicated syntax than the classical KN system of nested sequents presented in Chapter 1. Indeed, in the intuitionistic setting, although we need to have two-sided sequents, the succedent of a sequent is always reduced to a formula, and we have no need to generalise this. Such an intuitionistic nested sequent can therefore be thought of as a normal intuitionistic sequent, where a sequent from another branch is connected to one of the hypotheses in the antecedent, as follows, in the particular situation where the complete proof of A is grouped above at the bottom:

$$\begin{array}{c} \mathscr{P}\,\|\| \\ \Gamma, [\vdash B], C \vdash D \\ \mathscr{D}\,\|\| \\ \Gamma, [[\Delta \vdash A] \vdash B], C \vdash D \end{array} \quad \longrightarrow \quad \begin{array}{c} \mathscr{D} \quad\quad \mathscr{P} \\ \Delta \vdash A \qquad \Gamma, B, C \vdash D \\ \hline \\ \cdots \end{array}$$

1.2 A Family of Intuitionistic Proof Systems

The common basis for all systems[1] we present is called $\mathsf{JN} \cup \{\mathsf{e}\}$ and its inference rules are shown in Figure 1. It should be noticed that the *identity* i and *grab* g rules are exactly the same as in standard presentations of the sequent calculus [GLT89]. Then, the rules of *weakening* and *contraction*, w and c, are very similar but operate on a whole nested sequent, and not just a formula as in a shallow system. Finally, the rules of *cut* and *application*, called here e and a, rely on the use of nesting rather than branching. Moreover, both of them rely on the *switch* rule s, a specific feature of deep inference, to perform the context splitting required by their multiplicative presentation, in a lazy way. One switch instance moves only one sequent from an antecedent to the antecedent of another sequent — which is itself located in the same antecedent as the sequent being moved.

[1] Although the various systems we will present have structural differences, they share a basis of inference rules, using only variations of a small set of rules — notice that the names of some rules are chosen to recall their computational interpretation, but the grab and application rules will also be called right and left implication respectively, as in the sequent calculus.

$$\mathsf{i}\,\frac{}{A\vdash A} \qquad \mathsf{e}\,\frac{\Gamma,[A\vdash A]\vdash B}{\Gamma\vdash B} \qquad \mathsf{g}\,\frac{\Gamma,A\vdash B}{\Gamma\vdash A\to B} \qquad \mathsf{a}\,\frac{\Gamma,[\Delta,A\vdash B]\vdash C}{\Gamma,[\Delta\vdash A\to B]\vdash C}$$

$$\mathsf{w}\,\frac{\Gamma\vdash A}{\Gamma,\delta\vdash A} \qquad \mathsf{c}\,\frac{\Gamma,\delta,\delta\vdash A}{\Gamma,\delta\vdash A} \qquad \mathsf{s}\,\frac{\Gamma,[\Delta,[\Psi,\delta\vdash A]\vdash B]\vdash C}{\Gamma,\delta,[\Delta,[\Psi\vdash A]\vdash B]\vdash C}$$

Figure 1: Inference rules for the system $\mathsf{JN}\cup\{\mathsf{e}\}$

Example 1.4. *Here are proofs in* $\mathsf{JN}\cup\{\mathsf{e}\}$ *using all inference rules available. Notice that their conclusive sequent contain sequent nesteds in the left-hand side, so that they cannot be directly used as conclusive sequents in a usual sequent calculus system.*

$$\mathsf{g}\,\frac{\mathsf{c}\,\frac{\mathsf{s}\,\frac{\mathsf{i}\,\frac{\mathsf{i}\,\frac{}{B\vdash B}}{[[A\vdash A]\vdash B]\vdash B}}{[A\vdash B],A\vdash B}}{[A,A\vdash B],A,A\vdash B}}{[A,A\vdash B],A\vdash B}}{[A,A\vdash B]\vdash A\to B}$$

$$\mathsf{a}\,\frac{\mathsf{e}\,\frac{\mathsf{w}\,\frac{\mathsf{s}\,\frac{\mathsf{s}\,\frac{\mathsf{i}\,\frac{\mathsf{i}\,\frac{\mathsf{i}\,\frac{}{B\vdash B}}{[[B\vdash B]\vdash B]\vdash B}}{[[[[B\vdash B]\vdash B]\vdash B]\vdash B]\vdash B}}{[[B,[B\vdash B]\vdash B]\vdash B]\vdash B}}{B,[[[B\vdash B]\vdash B]\vdash B]\vdash B}}{B,[[[B\vdash B],A\vdash B]\vdash B]\vdash B}}{B,[[A\vdash B]\vdash B]\vdash B}}{B,[A\to B\vdash B]\vdash B}$$

We will use the identity rule in its general form and not only in atomic form, but it is always possible to reduce it to this particular situation, by replacing a general instance with a derivation using atomic identities, as shown below.

Proposition 1.5. *Any instance of the* i *rule can be replaced by a derivation in* JN *with same premise and conclusion, using instances of* i *only in the atomic form.*

Proof. We proceed by induction on the formula A affected by a general instance of the identity rule, with premise $\xi\{\,\}$ and conclusion $\xi\{A\vdash A\}$. If A is an atom a, then this identity is already in atomic form and we are done. In the general case, A is an implication $B\to C$, and we replace the initial instance by the following derivation:

$$\mathsf{g}\,\frac{\mathsf{a}\,\frac{\mathsf{s}\,\frac{\mathsf{i}\,\frac{\mathsf{i}\,\frac{\xi\{\,\}}{\xi\{C\vdash C\}}}{\xi\{[[B\vdash B]\vdash C]\vdash C\}}}{\xi\{[B\vdash C],B\vdash C\}}}{\xi\{B\to C,B\vdash C\}}}{\xi\{B\to C\vdash B\to C\}}$$

to which we can apply the induction hypothesis. □

There are many possible variations of the given set of inference rules, since we can generalise or restrict them, while retaining soundness and completeness with respect to intuitionistic logic, and we can also merge some of them so as to restrict the possible shapes of a proof. We list now some variants of inference rules that we have at our disposal to build different proof systems based on $\mathsf{JN} \cup \{\mathsf{e}\}$:

1. The identity rule iw with built-in weakening, typical of the sequent calculus, which allows to remove the weakening rule from the system, thus restricting the possible shapes of proofs — it is equivalent to a derivation using several weakenings and an identity:

$$\mathsf{iw}\ \frac{}{\Gamma, A \vdash A}$$

2. The cut rule es with built-in switch, which is very close to the cut used in the sequent calculus, and allows to remove the switch rule if it is also built in the a rule — it is equivalent to a derivation using a cut e and several switches:

$$\mathsf{es}\ \frac{\Gamma, [[\Delta \vdash A] \vdash A] \vdash B}{\Gamma, \Delta \vdash B}$$

3. The application rule as with built-in switch, which is a generalisation of the usual left implication rule of the sequent calculus, and allows to remove the switch when used together with the es rule — it is equivalent to a derivation using an application a and several switches:

$$\mathsf{as}\ \frac{\Gamma, [\Psi, [\Delta \vdash A] \vdash B] \vdash C}{\Gamma, \Delta, [\Psi \vdash A \to B] \vdash C}$$

4. The additive switch rule sa, which performs additive context distribution, and could also be obtained by using a general contraction on multisets of sequents as well as a generalised switch — it is equivalent to a derivation using several contractions and several switches:

$$\mathsf{sa}\ \frac{\Gamma, [\Delta, [\Psi, \Gamma \vdash A] \vdash B] \vdash C}{\Gamma, [\Delta, [\Psi \vdash A] \vdash B] \vdash C}$$

5. The cut rule esa with built-in additive switch, which resembles the additive cut of the sequent calculus and is a generalisation of es — it is equivalent to a derivation using a cut e and an additive switch:

$$\mathsf{esa}\ \frac{\Gamma, [[\Gamma \vdash A] \vdash A] \vdash B}{\Gamma \vdash B}$$

6. The application rule asa with built-in additive switch, which produces a proof system in additive style, if used together with the esa, iw and g rules — it is equivalent to a derivation using an application a and an additive switch:

$$\mathsf{asa}\ \frac{\Gamma, [\Delta, [\Gamma \vdash A] \vdash B] \vdash C}{\Gamma, [\Delta \vdash A \to B] \vdash C}$$

7. The blended cut rule ebs with built-in switch and contraction, the result of blending a multiplicative cut and an additive cut [Hug10] — it is equivalent to a derivation using a cut, a contraction a switch and an additive switch:

$$\mathsf{ebs}\,\frac{\Gamma, \Delta, [[\Delta, \Psi \vdash A] \vdash A] \vdash B}{\Gamma, \Delta, \Psi \vdash B}$$

8. The blended application rule abs with built-in switch and contraction, which is also a blend of multiplicative and additive style, and produces a system without contraction rule where the weakening is only needed on parts of the conclusive sequents that are irrelevant — it is equivalent to a derivation using an application a, a contraction, several switches s and an additive switch:

$$\mathsf{abs}\,\frac{\Gamma, \Delta, [\Sigma, [\Delta, \Psi \vdash A] \vdash B] \vdash C}{\Gamma, \Delta, \Psi, [\Sigma \vdash A \to B] \vdash C}$$

9. The restricted weakening rule wr, which only allows to weaken formulas, and not whole sequents, so that it corresponds to the weakening in the sequent calculus — it is equivalent to a normal weakening applied on a sequent of the shape $\vdash A$, with empty antecedent:

$$\mathsf{wr}\,\frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

10. The restricted contraction rule cr, which is also allowing to contract formulas only, and thus corresponds to the contraction in the sequent calculus — it is equivalent to a normal contraction on a sequent with empty antecedent:

$$\mathsf{cr}\,\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}$$

11. The restricted application rule ar, which only allows to decompose a formula on the right-hand side of a sequent with an empty antecedent, thus restricting the shape of proofs, so that branches can easily be seen — it is equivalent to a normal application a on a sequent with no antecedent:

$$\mathsf{ar}\,\frac{\Gamma, [A \vdash B] \vdash C}{\Gamma, A \to B \vdash C}$$

12. The cut rule ers with built-in restricted switch, which provides a splitting such that only formulas are placed in the new sequent, and is thus closer to the sequent calculus — it is equivalent to a derivation using a cut e and several switches in restricted form:

$$\mathsf{ers}\,\frac{\Gamma, [[F_1, \cdots, F_n \vdash A] \vdash A] \vdash B}{\Gamma, F_1, \cdots, F_n \vdash B}$$

13. The restricted application rule ars with built-in restricted switch, which splits the antecedent so that only formulas are moved inside, and is thus closer to the sequent calculus — it is equivalent to a derivation using an application a and several switches in restricted form:

$$\mathsf{ars}\ \frac{\Gamma, [[F_1, \cdots, F_n \vdash A] \vdash B] \vdash C}{\Gamma, F_1, \cdots, F_n, A \to B \vdash C}$$

14. The distant switch rule sdc with built-in contraction, allowing multiple copies of a sequent to be moved to an arbitrary depth in a context with several holes — it is equivalent to a derivation using several contractions and switches:

$$\mathsf{sdc}\ \frac{\Gamma, \zeta\{\Delta_i, \delta \vdash A_i\}^+ \vdash B}{\Gamma, \delta, \zeta\{\Delta_i \vdash A_i\}^+ \vdash B}$$

Of course, this list is not exhaustive, but it covers all the rules we will use in our systems, and those we will mention later. It is important to notice that all of them are derivable from rules of $\mathsf{JN} \cup \{\mathsf{e}\}$, or are special cases of these.

We can now pick different selections of these inference rule variations to build our proof systems. We produce this way four new proof systems, using different features provided by the variants of basic inference rules we listed above.

Definition 1.6. *Our proof system family for intuitionistic logic is defined as follows:*

$$\begin{aligned}
\mathsf{JN} \cup \{\mathsf{e}\} &= \{\mathsf{i}, \mathsf{e}, \mathsf{g}, \mathsf{a}, \mathsf{w}, \mathsf{c}, \mathsf{s}\} && \textit{(basic)}\\
\mathsf{JNs} \cup \{\mathsf{es}\} &= \{\mathsf{i}, \mathsf{es}, \mathsf{g}, \mathsf{as}, \mathsf{w}, \mathsf{c}\} && \textit{(no-switch)}\\
\mathsf{JNr} \cup \{\mathsf{ers}\} &= \{\mathsf{i}, \mathsf{ers}, \mathsf{g}, \mathsf{ars}, \mathsf{wr}, \mathsf{cr}\} && \textit{(restricted)}\\
\mathsf{JNa} \cup \{\mathsf{esa}\} &= \{\mathsf{iw}, \mathsf{esa}, \mathsf{g}, \mathsf{asa}\} && \textit{(additive)}\\
\mathsf{JNb} \cup \{\mathsf{ebs}\} &= \{\mathsf{i}, \mathsf{ebs}, \mathsf{g}, \mathsf{abs}, \mathsf{w}\} && \textit{(blended)}
\end{aligned}$$

These systems differ in the way they restrict the possible shapes of a proof, but can be compared, in the sense that some variant of inference rules can be used to simulate other variants. We can define this way a relation on proof systems.

Example 1.7. *Here are variations of the proofs given in Example 1.4, in the* $\mathsf{JNa} \cup \{\mathsf{esa}\}$ *and* $\mathsf{JNb} \cup \{\mathsf{ebs}\}$ *systems respectively, illustrating the way variant rules are used.*

$$\mathsf{g}\ \frac{\mathsf{asa}\ \frac{\mathsf{asa}\ \frac{\mathsf{iw}\ \frac{\mathsf{iw}\ \frac{}{B, A \vdash B}}{[[A \vdash A] \vdash B], A \vdash B}}{[[A \vdash A], [A \vdash A] \vdash B], A \vdash B}}{[[A \vdash A] \vdash A \to B], A \vdash B}}{A \to A \to B, A \vdash B}}{A \to A \to B \vdash A \to B}$$

$$\mathsf{abs}\ \frac{\mathsf{ebs}\ \frac{\mathsf{w}\ \frac{\mathsf{i}\ \frac{\mathsf{i}\ \frac{\mathsf{i}\ \frac{}{B \vdash B}}{[[B \vdash B] \vdash B] \vdash B}}{[[[[B \vdash B] \vdash B] \vdash B] \vdash B] \vdash B}}{[[[[B \vdash B] \vdash B] \vdash B], A \vdash B] \vdash B}}{[[B \vdash B], A \vdash B] \vdash B}}{[[B \vdash B] \vdash A \to B] \vdash B}$$

Definition 1.8. *The* inclusion *relation $\subseteq$ on proof systems is defined as the smallest reflexive, transitive relation such that, given two systems R and S, $R \subseteq S$ if and only if for any rule instance of R there is a derivation in S with same premise and conclusion.*

The basic $\mathsf{JN} \cup \{\mathsf{e}\}$ system is clearly the most general one, so that other systems are included in it, since all rule variants we use are compositions or restriction of its rules. The other systems are mainly organised around two branches, depending on the design choice adopted, either restriction or compound rules.

Proposition 1.9. *The inclusion relations between the various intuitionistic systems in nested sequents are the following:*

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{JNa} \cup \{\mathsf{esa}\} & \subseteq & \mathsf{JNb} \cup \{\mathsf{ebs}\} & \mathsf{JNb} \cup \{\mathsf{ebs}\} & \subseteq & \mathsf{JNs} \cup \{\mathsf{es}\} \\ \mathsf{JNr} \cup \{\mathsf{ers}\} & \subseteq & \mathsf{JNs} \cup \{\mathsf{es}\} & \mathsf{JNs} \cup \{\mathsf{es}\} & \subseteq & \mathsf{JN} \cup \{\mathsf{e}\} \end{array}$$

Proof. The proof is straightforward, since for a relation $R \subseteq S$ to be proven, we can provide for each rule of R an equivalent derivation of S, following the instructions given in the list of inference rule variations above. □

Remark 1.10. *An illustration of the different levels of expressiveness of systems in this hierarchy is that the sequent $[A, A \vdash B] \vdash A \to B$ for which a proof in the* JN *system is given in Example 1.4 cannot be proven in the* JNa *system. This is the reason why the corresponding proof in* JNa *given in Example 1.7 is a proof of the logically equivalent sequent $A \to A \to B \vdash A \to B$.*

With these definitions, we can discuss properties of these variations of $\mathsf{JN} \cup \{\mathsf{e}\}$, starting with the comparison with the sequent calculus for intuitionistic logic.

1.3 Correspondence to the Sequent Calculus

In order to show that all of our systems are suitable for intuitionistic logic, we have to prove soundness and completeness with respect to the sequent calculus. We use the variant shown in Figure 2, similar to standard presentations [GLT89], that we will simply call here $\mathsf{LJ} \cup \{\mathsf{cut}\}$. For that we need to translate nested sequents into formulas, since antecedents can only contain formulas in the sequent calculus.

Definition 1.11. *The translation $[\![\cdot]\!]_{\mathsf{F}}$ from nested sequents to intuitionistic formulas is defined recursively as follows:*

$$[\![\vdash A]\!]_{\mathsf{F}} = A \qquad \textit{and} \qquad [\![\delta, \Gamma \vdash A]\!]_{\mathsf{F}} = [\![\delta]\!]_{\mathsf{F}} \to [\![\Gamma \vdash A]\!]_{\mathsf{F}}$$

Remark 1.12. *We are only translating nested sequents into formulas[2], and thus what we show can be called* formula-completeness *[Hug10] and only ensures completeness when it comes to sequents of the shape $\vdash A$ and not sequents of any shape.*

Then, we can prove soundness with respect to intuitionistic logic, starting with the basic $\mathsf{JN} \cup \{\mathsf{e}\}$ proof system, since this is the most generic in the family, from the viewpoint of inference rules decomposition. For that we need to prove the three following technical lemmas, to simplify the proof of the theorem.

[2] A closer translation from nested to shallow sequents is not possible since nesting corresponds to branching, so that we would need to translate to a set of shallow sequents.

$$\mathsf{ax}\ \frac{}{A \vdash A} \qquad \mathsf{cut}\ \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B} \qquad \mathsf{weak}\ \frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

$$\rightarrow_{\mathsf{R}}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \qquad \rightarrow_{\mathsf{L}}\ \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \rightarrow B \vdash C} \qquad \mathsf{cont}\ \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}$$

Figure 2: Inference rules for system $\mathsf{LJ} \cup \{\mathsf{cut}\}$

Lemma 1.13. *For any antecedents Γ, Δ, Ψ, and any formulas A, B, if the implication $[\![\Delta \vdash A]\!]_{\mathsf{F}} \rightarrow [\![\Psi \vdash B]\!]_{\mathsf{F}}$ is provable in LJ, then $[\![\Gamma, \Delta \vdash A]\!]_{\mathsf{F}} \rightarrow [\![\Gamma, \Psi \vdash B]\!]_{\mathsf{F}}$ is also provable in LJ.*

Proof. By induction on the number of elements in Γ. If Γ is empty, then the result is trivial. In the general case, we consider one element δ in Γ, so that $\Gamma = \delta, \Gamma'$ and by induction hypothesis the implication $[\![\Gamma', \Delta \vdash A]\!]_{\mathsf{F}} \rightarrow [\![\Gamma', \Psi \vdash B]\!]_{\mathsf{F}}$ is provable in LJ by some proof Π. We can build the following proof LJ:

$$\rightarrow_{\mathsf{R}}{}^{*}\ \frac{\rightarrow_{\mathsf{L}}\ \dfrac{\mathsf{ax}\ \dfrac{}{[\![\delta]\!]_{\mathsf{F}} \vdash [\![\delta]\!]_{\mathsf{F}}} \qquad \overset{\Pi'}{\triangledown}\ [\![\Gamma', \Delta \vdash A]\!]_{\mathsf{F}} \vdash [\![\Gamma', \Psi \vdash B]\!]_{\mathsf{F}}}{[\![\delta]\!]_{\mathsf{F}} \rightarrow [\![\Gamma', \Delta \vdash A]\!]_{\mathsf{F}}, [\![\delta]\!]_{\mathsf{F}} \vdash [\![\Gamma', \Psi \vdash B]\!]_{\mathsf{F}}}}{\vdash ([\![\delta]\!]_{\mathsf{F}} \rightarrow [\![\Gamma', \Delta \vdash A]\!]_{\mathsf{F}}) \rightarrow [\![\delta]\!]_{\mathsf{F}} \rightarrow [\![\Gamma', \Psi \vdash B]\!]_{\mathsf{F}}}$$

where the proof Π' is obtained from Π by invertibility of the $\rightarrow_{\mathsf{R}}$ rule, which is a well-known result in the sequent calculus, and this is indeed the expected proof in LJ, by definition of the $[\![\cdot]\!]_{\mathsf{F}}$ translation. □

Lemma 1.14. *For any antecedents Γ, Δ, and any formulas A, B, C, if the implication $[\![\Gamma \vdash A]\!]_{\mathsf{F}} \rightarrow [\![\Delta \vdash B]\!]_{\mathsf{F}}$ is provable in the LJ system, then we can also prove that the implication $[\![[\Delta \vdash B] \vdash C]\!]_{\mathsf{F}} \rightarrow [\![[\Gamma \vdash A] \vdash C]\!]_{\mathsf{F}}$ holds in LJ.*

Proof. Assuming we have a proof Π of $[\![\Gamma \vdash A]\!]_{\mathsf{F}} \rightarrow [\![\Delta \vdash B]\!]_{\mathsf{F}}$, we can easily obtain a proof Π' of $[\![\Gamma \vdash A]\!]_{\mathsf{F}} \vdash [\![\Delta \vdash B]\!]_{\mathsf{F}}$, again by invertibility of the $\rightarrow_{\mathsf{R}}$ rule. We use it to build the following proof:

$$\rightarrow_{\mathsf{R}}{}^{*}\ \frac{\rightarrow_{\mathsf{L}}\ \dfrac{\overset{\Pi'}{\triangledown}\ [\![\Gamma \vdash A]\!]_{\mathsf{F}} \vdash [\![\Delta \vdash B]\!]_{\mathsf{F}} \qquad \mathsf{ax}\ \dfrac{}{C \vdash C}}{[\![\Delta \vdash B]\!]_{\mathsf{F}} \rightarrow C, [\![\Gamma \vdash A]\!]_{\mathsf{F}} \vdash C}}{\vdash ([\![\Delta \vdash B]\!]_{\mathsf{F}} \rightarrow C) \rightarrow [\![\Gamma \vdash A]\!]_{\mathsf{F}} \rightarrow C}$$

which is the expected proof in LJ, by definition of the $[\![\cdot]\!]_{\mathsf{F}}$ translation. □

In the previous lemmas, we consider that the empty nested sequent behaves as a unit for the implication, so that for example Lemma 1.14 can be used to reduce the provability of the formula $[\![[\Delta \vdash B] \vdash C]\!]_{\mathsf{F}} \to C$ to the provability of $[\![\Delta \vdash B]\!]_{\mathsf{F}}$, since C is exactly $[\![\vdash C]\!]_{\mathsf{F}}$ by definition of the $[\![\cdot]\!]_{\mathsf{F}}$ translation.

Lemma 1.15. *For any antecedents Γ, Δ, context ξ, sequent δ and formulas A and B, if the implication $[\![\Gamma, \delta \vdash A]\!]_{\mathsf{F}} \to [\![\Delta \vdash B]\!]_{\mathsf{F}}$ is provable in LJ, then we can also prove the implication $[\![\Gamma, \xi\{\delta\} \vdash A]\!]_{\mathsf{F}} \to [\![\Delta, \xi\{\ \} \vdash B]\!]_{\mathsf{F}}$ in LJ.*

Proof. In any proof Π of $[\![\Gamma, \delta \vdash A]\!]_{\mathsf{F}} \to [\![\Delta \vdash B]\!]_{\mathsf{F}}$ in LJ, there exists necessarily a subproof Π_1 of a sequent of the shape $\Psi \vdash [\![\delta]\!]_{\mathsf{F}}$. Then, we can build a proof Π_1' of the sequent $\Psi, [\![\xi\{\ \}]\!]_{\mathsf{F}} \vdash [\![\xi\{\delta\}]\!]_{\mathsf{F}}$ using a straightforward induction on ξ. We can thus build the resulting proof by replacing Π_1 with Π_1' and rewriting δ into $\xi\{\delta\}$ everywhere in the proof Π. □

Now, we use the standard technique for proving soundness of a deep inference proof system, by showing that rule instances correspond to valid implications, and using cuts to build a proof in the sequent calculus. Notice that to prove soundness for cut-free JN using this technique, we would need the cut elimination result.

Theorem 1.16 (Soundness of $\mathsf{JN} \cup \{\mathsf{e}\}$)**.** *If a nested sequent κ is provable in $\mathsf{JN} \cup \{\mathsf{e}\}$, then the formula $[\![\kappa]\!]_{\mathsf{F}}$ is provable in the $\mathsf{LJ} \cup \{\mathsf{cut}\}$ sequent calculus.*

Proof. We proceed by induction on the length of a proof $\mathscr{D}$ of κ in $\mathsf{JN} \cup \{\mathsf{e}\}$. In the base case, when $\mathscr{D}$ is just one rule instance, it has to be an instance of i, applied on a sequent of the form $A \vdash A$, and we are done since $[\![A \vdash A]\!]_{\mathsf{F}} = A \to A$ is provable in $\mathsf{LJ} \cup \{\mathsf{cut}\}$. In the general case, we consider the bottommost instance r in $\mathscr{D}$:

$$\mathsf{r}\,\dfrac{\begin{array}{c}\Vdots\\ \xi\{\mu\}\end{array}}{\xi\{\upsilon\}}$$

We have to show first that the implication $[\![\mu]\!]_{\mathsf{F}} \to [\![\upsilon]\!]_{\mathsf{F}}$ is provable in LJ, and for this we use a case analysis on r and build a proof Π_1 of this implication:

1. If r is an instance of i, we have to prove again $A \to A$ for some formula A and we simply use the $\to_{\mathsf{R}}$ rule and the axiom rule again.
2. If r is an instance of e, we can use Lemma 1.13 and then Lemma 1.14 to reduce the provability of $[\![\mu]\!]_{\mathsf{F}} \to [\![\upsilon]\!]_{\mathsf{F}}$ to the provability of $A \to A$, and we use $\to_{\mathsf{R}}$ and the axiom rule, as in the case of the identity.
3. If r is a w instance, we reduce by Lemma 1.13 the problem to the provability of the formula $A \to [\![\delta]\!]_{\mathsf{F}} \to A$, for which we build the following proof:

$$\to_{\mathsf{R}}{}^{*}\,\dfrac{\mathsf{weak}\,\dfrac{\mathsf{ax}\,\dfrac{}{A \vdash A}}{A, [\![\delta]\!]_{\mathsf{F}} \vdash A}}{\vdash A \to [\![\delta]\!]_{\mathsf{F}} \to A}$$

4. If r is a c instance, we reduce by Lemma 1.13 the problem to the provability of the formula $(\llbracket\delta\rrbracket_{\mathsf{F}} \to \llbracket\delta\rrbracket_{\mathsf{F}} \to A) \to \llbracket\delta\rrbracket_{\mathsf{F}} \to A$, for which we build the following proof:

$$\dfrac{\dfrac{\dfrac{\mathsf{ax}\,\dfrac{}{\llbracket\delta\rrbracket_{\mathsf{F}} \vdash \llbracket\delta\rrbracket_{\mathsf{F}}} \quad \dfrac{\mathsf{ax}\,\dfrac{}{\llbracket\delta\rrbracket_{\mathsf{F}} \vdash \llbracket\delta\rrbracket_{\mathsf{F}}} \quad \mathsf{ax}\,\dfrac{}{A \vdash A}}{\llbracket\delta\rrbracket_{\mathsf{F}} \to A, \llbracket\delta\rrbracket_{\mathsf{F}} \vdash A}\,{\to_{\mathsf{L}}}}{\llbracket\delta\rrbracket_{\mathsf{F}} \to \llbracket\delta\rrbracket_{\mathsf{F}} \to A, \llbracket\delta\rrbracket_{\mathsf{F}}, \llbracket\delta\rrbracket_{\mathsf{F}} \vdash A}\,{\to_{\mathsf{L}}}}{\llbracket\delta\rrbracket_{\mathsf{F}} \to \llbracket\delta\rrbracket_{\mathsf{F}} \to A, \llbracket\delta\rrbracket_{\mathsf{F}} \vdash A}\,\mathsf{cont}}{\vdash (\llbracket\delta\rrbracket_{\mathsf{F}} \to \llbracket\delta\rrbracket_{\mathsf{F}} \to A) \to \llbracket\delta\rrbracket_{\mathsf{F}} \to A}\,{\to_{\mathsf{R}}^{*}}$$

5. If r is a g instance, we reduce by Lemma 1.13 the problem to the provability of the formula $(A \to B) \to A \to B$, which is trivial using the axiom rule.

6. If r is an a instance, we can use Lemma 1.13 and Lemma 1.14 to reduce the problem to the provability of the formula $(A \to B) \to A \to B$, which is trivial.

7. If r is an instance of s, we can use Lemma 1.15 to reduce the problem to the provability of $\llbracket\Gamma, \delta \vdash C\rrbracket_{\mathsf{F}} \to \llbracket\Gamma, \delta \vdash C\rrbracket_{\mathsf{F}}$, which is trivial, since in the switch rule the sequent $[\Delta, [\Psi, \{\,\} \vdash A] \vdash B]$ is a context that is not modified.

Then, given a context ξ, we can prove by a straightforward induction on ξ, and by invertibility of the $\to_{\mathsf{R}}$ rule, that there is a proof Π_2 of $\llbracket\xi\{\mu\}\rrbracket_{\mathsf{F}} \vdash \llbracket\xi\{\upsilon\}\rrbracket_{\mathsf{F}}$ in $\mathsf{LJ} \cup \{\mathsf{cut}\}$. By induction hypothesis, we also have a proof Π_3 of $\llbracket\xi\{\mu\}\rrbracket_{\mathsf{F}}$, and thus we can finally build a proof of $\llbracket\xi\{\upsilon\}\rrbracket_{\mathsf{F}}$ by using a cut as follows:

$$\mathsf{cut}\,\dfrac{\vdash \llbracket\xi\{\mu\}\rrbracket_{\mathsf{F}} \quad \llbracket\xi\{\mu\}\rrbracket_{\mathsf{F}} \vdash \llbracket\xi\{\upsilon\}\rrbracket_{\mathsf{F}}}{\vdash \llbracket\xi\{\upsilon\}\rrbracket_{\mathsf{F}}}$$

where the premise $\vdash \llbracket\xi\{\mu\}\rrbracket_{\mathsf{F}}$ is given by Π_3 and the premise $\llbracket\xi\{\mu\}\rrbracket_{\mathsf{F}} \vdash \llbracket\xi\{\upsilon\}\rrbracket_{\mathsf{F}}$ is given by Π_2. □

This result can be immediately extended to all the other systems of the family, since they are included in $\mathsf{JN} \cup \{\mathsf{e}\}$ and thus a proof in one of these system can be simulated by a proof in $\mathsf{JN} \cup \{\mathsf{e}\}$.

Corollary 1.17. *The proof systems* $\mathsf{JNa} \cup \{\mathsf{esa}\}$, $\mathsf{JNb} \cup \{\mathsf{ebs}\}$, $\mathsf{JNr} \cup \{\mathsf{ers}\}$, $\mathsf{JNs} \cup \{\mathsf{es}\}$ *are sound with respect to the sequent calculus* $\mathsf{LJ} \cup \{\mathsf{cut}\}$ *for intuitionistic logic.*

Proof. For any proof $\mathscr{D}$ of a sequent δ in one of these systems, there is a proof $\mathscr{D}'$ of δ in $\mathsf{JN} \cup \{\mathsf{e}\}$, since this system can simulate all the other systems of the family, as shown in Proposition 1.9, and thus by Theorem 1.16 there is a proof of the formula $\llbracket\delta\rrbracket_{\mathsf{F}}$ in the sequent calculus $\mathsf{LJ} \cup \{\mathsf{cut}\}$. □

In order to prove completeness for these systems with respect to intuitionistic logic, we prove the existence of the opposite translation, and we do this for a weak proof system, so that it can be extended to other systems.

Theorem 1.18 (Completeness of $\mathsf{JNr} \cup \{\mathsf{ers}\}$)**.** *If a shallow sequent $\Sigma \vdash F$ is provable in $\mathsf{LJ} \cup \{\mathsf{cut}\}$, then the nested sequent $\Sigma \vdash F$ is provable in $\mathsf{JNr} \cup \{\mathsf{ers}\}$.*

Proof. By induction on a proof Π of the sequent $\Sigma \vdash F$ in $\mathsf{LJ} \cup \{\mathsf{cut}\}$, using a case analysis on the bottommost rule instance r in Π, we build a proof $\mathscr{D}$ of the nested translation of this sequent in the $\mathsf{JNr} \cup \{\mathsf{ers}\}$ system:

1. If r is an instance of ax, we can simply use an identity rule, and we reached the initial case of the induction, so that we are done:

$$\mathsf{ax}\,\frac{}{A \vdash A} \qquad \longrightarrow \qquad \mathsf{iw}\,\frac{}{A \vdash A}$$

2. If r is an instance of cut, then we produce the proofs $\mathscr{D}_1$ and $\mathscr{D}_2$ in $\mathsf{JNr} \cup \{\mathsf{ers}\}$ by applying the induction hypothesis to the premises Π_1 and Π_2 of the cut instance, and we can build the resulting proof using the ers rule because Γ and Δ are shallow antecedents, which contain only formulas, since they were obtained by translation of a shallow sequent:

$$\mathsf{cut}\,\frac{\overset{\Pi_1}{\Gamma \vdash A} \qquad \overset{\Pi_2}{\Delta, A \vdash B}}{\Gamma, \Delta \vdash B} \qquad \longrightarrow \qquad \mathsf{ers}\,\frac{\begin{array}{c}\overset{\mathscr{D}_2}{\Gamma, [\vdash A] \vdash B} \\ \overset{\mathscr{D}_1'}{\Gamma, [[\Delta \vdash A] \vdash A] \vdash B}\end{array}}{\Gamma, \Delta \vdash B}$$

   where $\mathscr{D}_1'$ is obtained from $\mathscr{D}_1$ by plugging it into the context $\Gamma, [\{\,\} \vdash A] \vdash B$, which is possible because of the deep inference methodology.

3. If r is an instance of weak, we apply the induction hypothesis to the premise Π_1 of the weakening to produce a proof $\mathscr{D}_1$, and we can build the resulting proof using wr since only a formula is affected:

$$\mathsf{weak}\,\frac{\overset{\Pi_1}{\Gamma \vdash B}}{\Gamma, A \vdash B} \qquad \longrightarrow \qquad \mathsf{wr}\,\frac{\overset{\mathscr{D}_1}{\Gamma \vdash B}}{\Gamma, A \vdash B}$$

4. If r is an instance of cont, we apply the induction hypothesis to the premise Π_1 of the contraction to produce a proof $\mathscr{D}_1$, and we can build the resulting proof using cr since only a formula is affected:

$$\mathsf{cont}\,\frac{\overset{\Pi_1}{\Gamma, A, A \vdash B}}{\Gamma, A \vdash B} \qquad \longrightarrow \qquad \mathsf{cr}\,\frac{\overset{\mathscr{D}_1}{\Gamma, A, A \vdash B}}{\Gamma, A \vdash B}$$

5. If r is an instance of $\rightarrow_{\mathsf{R}}$, we apply the induction hypothesis to the premise Π_1 of this instance to produce a proof $\mathscr{D}_1$, and we build the result using g:

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A \rightarrow B}\,{\scriptstyle \rightarrow_{\mathsf{R}}} \quad\longrightarrow\quad \dfrac{\begin{array}{c}\mathscr{D}_1\Big\Vert\\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A \rightarrow B}\,{\scriptstyle \mathsf{g}}$$

6. If r is an instance of $\rightarrow_{\mathsf{L}}$, then we produce the proofs $\mathscr{D}_1$ and $\mathscr{D}_2$ by applying the induction hypothesis to the premises Π_1 and Π_2 of this instance, and we can build the resulting proof using the ars rule because Γ and Δ are shallow antecedents, since they come from a shallow sequent:

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma \vdash A\end{array} \qquad \begin{array}{c}\Pi_2\\ \Delta, B \vdash C\end{array}}{\Gamma, \Delta, A \rightarrow B \vdash C}\,{\scriptstyle \rightarrow_{\mathsf{L}}} \quad\longrightarrow\quad \dfrac{\begin{array}{c}\mathscr{D}_2\Big\Vert\\ \Delta, [\vdash B] \vdash C\end{array}}{\Gamma, \Delta, [\vdash A \rightarrow B] \vdash C}\,{\scriptstyle \mathsf{ars}}$$

where again, $\mathscr{D}_1'$ is obtained from $\mathscr{D}_1$ by plugging this proof into the context $\Delta, [\{\,\} \vdash B] \vdash C$, using the deep inference features of the nested sequents. □

Corollary 1.19. *The proof systems* $\mathsf{JNs} \cup \{\mathsf{es}\}$ *and* $\mathsf{JN} \cup \{\mathsf{e}\}$ *are complete with respect to the sequent calculus* $\mathsf{LJ} \cup \{\mathsf{cut}\}$ *for intuitionistic logic.*

Proof. For any proof Π of a sequent $\Sigma \vdash F$ in $\mathsf{LJ} \cup \{\mathsf{cut}\}$ there is by Theorem 1.18 a proof $\mathscr{D}$ of this sequent in $\mathsf{JNr} \cup \{\mathsf{ers}\}$, and thus there is also a proof of this sequent in $\mathsf{JNs} \cup \{\mathsf{es}\}$ and $\mathsf{JN} \cup \{\mathsf{e}\}$, since they can simulate any proof of $\mathsf{JNr} \cup \{\mathsf{ers}\}$. □

Now, we have to give another proof of completeness for the additive system and the blended system, since they cannot simulate $\mathsf{JNr} \cup \{\mathsf{ers}\}$. The new proof is very similar to the previous one, but we need two technical lemmas.

Lemma 1.20. *If there is a proof of* $\xi\{\Gamma \vdash B\}$ *in* $\mathsf{JNa} \cup \{\mathsf{esa}\}$*, then for any formula A, there is also a proof of* $\xi\{\Gamma, A \vdash B\}$ *in* $\mathsf{JNa} \cup \{\mathsf{esa}\}$.

Proof. By induction on the length of the given proof $\mathscr{D}$ of $\xi\{\Gamma \vdash B\}$ in $\mathsf{JNa} \cup \{\mathsf{esa}\}$, using a case analysis on the bottommost rule instance r in $\mathscr{D}$. If r is a matching iw instance, we replace it by an instance which also deletes A, and this is also the base case. In all other cases, we use the induction hypothesis, and if needed replace r by the same instance where A is added to the antecedent in the conclusion. □

Lemma 1.21. *If there is a proof of* $\xi\{\Gamma, A, A \vdash B\}$ *in the* $\mathsf{JNa} \cup \{\mathsf{esa}\}$ *system, then there for any formula A, there is also a proof of* $\xi\{\Gamma, A \vdash B\}$ *in* $\mathsf{JNa} \cup \{\mathsf{esa}\}$.

Proof. We proceed the same way as in the proof of Lemma 1.20, except in the case of a matching instance of iw, where the new instance deletes one less copy of A. □

Theorem 1.22 (Completeness of $\mathsf{JNa} \cup \{\mathsf{esa}\}$)**.** *If a given shallow sequent* $\Gamma \vdash A$ *is provable in* $\mathsf{LJ} \cup \{\mathsf{cut}\}$*, then the nested sequent* $\Gamma \vdash A$ *is provable in* $\mathsf{JNa} \cup \{\mathsf{esa}\}$.

Proof. By induction on a proof Π of the sequent $\Gamma \vdash A$ in $\mathsf{LJ} \cup \{\mathsf{cut}\}$, using a case analysis on the bottommost rule instance r in Π, we build a proof $\mathscr{D}$ of the nested translation of this sequent in the $\mathsf{JNa} \cup \{\mathsf{esa}\}$ system:

1. If r is an instance of ax, then we use an identity rule in the special situation where there is nothing to weaken, and we are done:

$$\mathsf{ax}\ \frac{}{A \vdash A} \qquad \longrightarrow \qquad \mathsf{iw}\ \frac{}{A \vdash A}$$

2. If r is an instance of cut, then we produce the proofs $\mathscr{D}_1$ and $\mathscr{D}_2$ in $\mathsf{JNa} \cup \{\mathsf{esa}\}$ by applying the induction hypothesis to the premises Π_1 and Π_2 of this cut instance, and build the result as follows:

$$\mathsf{cut}\ \frac{\overset{\Pi_1}{\Gamma \vdash A} \qquad \overset{\Pi_2}{\Delta, A \vdash B}}{\Gamma, \Delta \vdash B} \qquad \longrightarrow \qquad \mathsf{esa}\ \frac{\begin{array}{c}\overset{\mathscr{D}_2'}{\vdots} \\ \Gamma, \Delta, [\vdash A] \vdash B \\ \overset{\mathscr{D}_1'}{\vdots} \\ \Gamma, \Delta, [[\Gamma, \Delta \vdash A] \vdash A] \vdash B\end{array}}{\Gamma, \Delta \vdash B}$$

   where $\mathscr{D}_2'$ is obtained by repeatedly applying Lemma 1.20 to $\mathscr{D}_2$, and $\mathscr{D}_1'$ by applying it to $\mathscr{D}_1$ and plugging the result into the context $\Gamma, \Delta, [\{\,\} \vdash A] \vdash B$, using the deep inference features of nested sequents.

3. If r is an instance of weak, we apply the induction hypothesis to the premise Π_1 of the weakening and use Lemma 1.20 on the result $\mathscr{D}_1$, to obtain another proof $\mathscr{D}_1'$ that we use to build the resulting proof:

$$\mathsf{weak}\ \frac{\overset{\Pi_1}{\Gamma \vdash B}}{\Gamma, A \vdash B} \qquad \longrightarrow \qquad \begin{array}{c}\overset{\mathscr{D}_1'}{\vdots} \\ \Gamma, A \vdash B\end{array}$$

4. If r is an instance of cont, we apply the induction hypothesis to the premise Π_1 of the contraction and use Lemma 1.21 on the result $\mathscr{D}_1$, to obtain another proof $\mathscr{D}_1'$, that we use to build the resulting proof:

$$\mathsf{cont}\ \frac{\overset{\Pi_1}{\Gamma, A, A \vdash B}}{\Gamma, A \vdash B} \qquad \longrightarrow \qquad \begin{array}{c}\overset{\mathscr{D}_1'}{\vdots} \\ \Gamma, A \vdash B\end{array}$$

5. If r is an instance of $\to_{\mathsf{R}}$, we apply the induction hypothesis to the premise Π_1 of this instance to produce a proof $\mathscr{D}_1$, and we build the result using g:

$$\dfrac{\overset{\Pi_1}{\vdots}\ \Gamma, A \vdash B}{\to_{\mathsf{R}}\ \Gamma \vdash A \to B} \quad\longrightarrow\quad \mathsf{g}\ \dfrac{\overset{\mathscr{D}_1}{\Vdots}\ \Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

6. If r is an instance of $\to_{\mathsf{L}}$, then we produce the proofs $\mathscr{D}_1$ and $\mathscr{D}_2$ by applying the induction hypothesis to its premises Π_1 and Π_2, and build the resulting proof as follows:

$$\dfrac{\overset{\Pi_1}{\vdots}\ \Gamma \vdash A \qquad \overset{\Pi_2}{\vdots}\ \Delta, B \vdash C}{\to_{\mathsf{L}}\ \Gamma, \Delta, A \to B \vdash C} \quad\longrightarrow\quad \mathsf{asa}\ \dfrac{\dfrac{\overset{\mathscr{D}_1'}{\Vdots}\ \Gamma, \Delta, [\Gamma, \Delta \vdash A] \vdash B}{\overset{\mathscr{D}_2'}{\Vdots}\ \Gamma, \Delta, [\vdash B] \vdash C}}{\Gamma, \Delta, [\vdash A \to B] \vdash C}$$

where $\mathscr{D}_2'$ is obtained by repeatedly applying Lemma 1.20 to $\mathscr{D}_2$, and $\mathscr{D}_1'$ by also applying it to $\mathscr{D}_1$ and plugging the result into $\Gamma, \Delta, [\{\} \vdash B] \vdash C$. □

Corollary 1.23. *The proof system* $\mathsf{JNb} \cup \{\mathsf{ebs}\}$ *is complete with respect to the sequent calculus* $\mathsf{LJ} \cup \{\mathsf{cut}\}$ *for intuitionistic logic.*

Proof. For any proof Π of a sequent $\Sigma \vdash F$ in $\mathsf{LJ} \cup \{\mathsf{cut}\}$ there is by Theorem 1.22 a proof $\mathscr{D}$ of this sequent in $\mathsf{JNa} \cup \{\mathsf{esa}\}$, and thus there is also a proof of this sequent in $\mathsf{JNb} \cup \{\mathsf{ebs}\}$, since it can simulate any proof of $\mathsf{JNa} \cup \{\mathsf{esa}\}$. □

Note that all the cut-free systems JNr, JNa, JNb, JNs and JN are obviously complete with respect to the cut-free LJ sequent calculus, because the different cut rules are needed only when translating the cut rule of the sequent calculus. From this, and from the cut admissibility result in the sequent calculus, we immediately get a cut elimination result. However, in the next section we will show several cut elimination procedures internal to the systems.

2 Cut Elimination

In this section we will present several variations of a cut elimination procedure for the proof systems of the family defined in the previous section. They are close to the usual procedure used in the sequent calculus [Gal93] in the sense that cuts are moved up in the proof until they meet a matching identity rule. They are also all based on the same machinery, established by some important lemmas concerning rule instances permutations and rewritings, that we will combine in different ways to build the variations of the basic idea of the procedure. The differences might seem unimportant from a purely logical viewpoint, but they matter when looking for a computational account of such procedures [Her94, BG00] — if we only cared about cut admissibility, we could have use soundness and completeness.

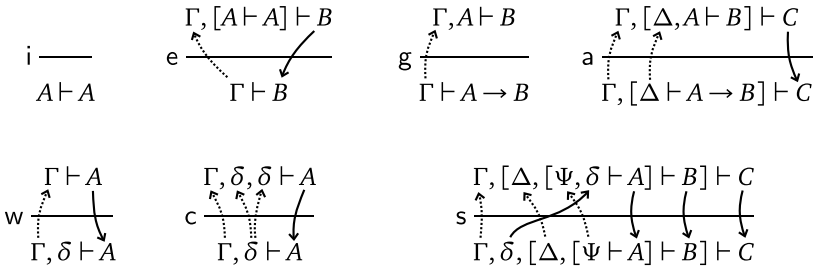
In the following, we will use generic results as much as possible, to be applied on several of our proof systems. To denote any of the systems defined above, we use the notation $\mathsf{JN}x \cup \{\mathsf{ex}\}$, and we handle all the possibilities in the case analyses used in the proofs. All variants of $\mathsf{JN} \cup \{\mathsf{e}\}$ are close enough to each other so that many cases can be factorised, making proofs easier to read.

2.1 Preliminaries

Most of the techniques we will use for cut elimination are based on permutations of rule instances, so that we will heavily rely on the definitions given in Chapter 1, concerning not only permutations, but also the formula and sequent particles and the flow-graph structure of a proof. As usual, we will denoted formula particles by $\underline{A}$ or $\underline{B}$ and so on, and sequent particles in the same way, by $\underline{\kappa}$ or $\underline{\delta}$, and so on.

Notice that, going up in a proof, any sequent particle can either be introduced by a cut, removed by an identity or by weakening, or duplicated by a contraction, so that other rules only move material into the antecedent of a sequent — when the structural rules are built inside the other rules, as done in the additive system $\mathsf{JNa} \cup \{\mathsf{esa}\}$ for example, the other rules have the same effect. This induces a direct correlation between the number of structural rule instances, or of instances with a structural feature, and the multiplicity of a particle or instance.

In the intuitionistic setting, we have two-sided sequents and we should thus be careful in the use of flow-graphs: there are two directions depending on the side of the sequent where a particle appears. However, we are mainly interested here in notion such as the multiplicity, defined regardless of directions. We can thus follow the intuition that to know a multiplicity, for example, we can put a finger on a each active particle, in a rule instance r, and follow their flow upwards in the derivation — if contractions are applied on the way, we need more fingers. In the $\mathsf{JN} \cup \{\mathsf{e}\}$ system, the connections inducing the flow-graph are the following — not all of them are shown, the connections between modified sequents are as expected:



To simplify the picture, one can consider the *negative flow-graph* $\mathscr{N}(\mathscr{D})$ of some derivation $\mathscr{D}$, where only particles in negative positions are considered, since these are the ones that can be directly affected by a contraction. Beyond the multiplicity $\mathscr{M}_{\mathscr{D}}(\underline{\kappa})$ of some particle $\underline{\kappa}$ in $\mathscr{D}$, we also need to consider the particles that appear in the flow of another particle, and specify cases where they have been modified by a rule instance.

Definition 2.1. *Given a particle $\underline{\kappa}$ in a derivation $\mathscr{D}$, an* ancestor *$\underline{\delta}$ of $\underline{\kappa}$ is a particle located in the flow of $\underline{\kappa}$ and above it in $\mathscr{D}$ — it is a* proper ancestor *if $\|\underline{\delta}\| \neq \|\underline{\kappa}\|$.*

The notion of ancestor can be lifted to the level of the inference rule instances used in some derivation, by simply looking at their active particles. This relation will be oriented upwards in the proof, as the basic manipulation to be performed on a rule instances is its permutation, which is usually done upwards — in particular, this is the case of cuts, for which the set of rules with some related active particles is important in the permutation process.

Definition 2.2. *In a derivation $\mathscr{D}$, the* scope *of a particle $\underline{\kappa}$ is the set of all the rule instances r above $\underline{\kappa}$ which have an active particle in the flow of $\underline{\kappa}$.*

This definition can be extended immediately to say that some rule instance r_1 in a derivation $\mathscr{D}$ is in the scope of another instance r_2 if there are particles $\underline{\kappa}$ and $\underline{\delta}$ active in r_1 and r_2 respectively, such that $\underline{\kappa}$ is in the scope of $\underline{\delta}$. The scope of a rule instance is thus the set of all rule instances above it in a derivation which are not » *independent* « from this instance, since they operate on sequents that were obtained by applying other rule instances to the active sequents of this instance.

Remark 2.3. *In the $\mathsf{JN} \cup \{\mathsf{e}\}$ system, only the contraction rule can create branches in the flow of a particle, and the duplicated particle is active in this contraction instance. Therefore, the multiplicity of a rule instance r can also be defined as the number of contraction instances in the scope of r. In systems where the contraction is built inside other rules, such as $\mathsf{JNa} \cup \{\mathsf{esa}\}$, these other rules must be counted as well.*

2.2 Weak Linearisation

The complexity of the cut elimination procedure is in a large part induced by the presence of contractions. Indeed, when the cut instances are moved up in a proof, they can meet contractions and thus be duplicated, so that the number of cuts in a proof is not a good measure for an induction. The *weak linearisation* process allows to transform any derivation so that contractions are applied at particular positions, making other proof transformations easier.

Definition 2.4. *Given any rule instance r, a derivation $\mathscr{D}$ in the $\mathsf{JNx} \cup \{\mathsf{ex}\}$ system is said to be* linear in r *if it is such that we have $\mathscr{M}_{\mathscr{D}}(\mathsf{r}) = 0$.*

This can be immediatly extended to say that some derivation $\mathscr{D}$ is linear in the particle $\underline{\kappa}$ when we have $\mathscr{M}_{\mathscr{D}}(\underline{\kappa}) = 0$. This leads us to the definition of the partially linearised form of proofs we are interested in.

Definition 2.5. *Given a derivation $\mathscr{D}$ in $\mathsf{JNx} \cup \{\mathsf{ex}\}$ and $\underline{\kappa}$ one of its sequent particles, $\mathscr{D}$ is said to be* weakly linear in *$\underline{\kappa}$ if it is linear in all proper ancestor of $\underline{\kappa}$.*

We will see that it is not always possible to make a given derivation linear in a given particle, but we can make it weakly linear in *sequent* particles. In other words, we can use contraction in a derivation $\mathscr{D}$ on sources of $\mathscr{N}(\mathscr{D})$ only — another way to see this is that we can make any rule instance in $\mathscr{D}$ linear.

Definition 2.6. *In a derivation $\mathscr{D}$ in* $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$*, a rule instance* r *is* non-weak *for a sequent particle $\underline{\kappa}$ if it is in the scope of a proper ancestor of $\underline{\kappa}$.*

The idea is that in a given derivation $\mathscr{D}$, the multiplicity of some source of $\mathscr{N}(\mathscr{D})$ cannot always decrease by permuting down contractions, but it can decrease for other particles, for which there are non-weak contractions in the derivation. Notice that this terminology of *weak linearity* — where linear is understood as *affine*, since weakenings are not moved — corresponds to the one used in the λ-calculus [AF05].

Remark 2.7. *The weak linearisation of a sequent in a derivation relies on permuting contractions down under other instances, which cannot be done in the restricted system* $\mathsf{JN}\mathsf{r} \cup \{\mathsf{ers}\}$ *since it would require the general form of the contraction rule.*

Lemma 2.8. *For any derivation $\mathscr{D}$ in one of the proof systems* $\mathsf{JN} \cup \{\mathsf{e}\}$*,* $\mathsf{JNs} \cup \{\mathsf{es}\}$*,* $\mathsf{JNa} \cup \{\mathsf{esa}\}$ *and* $\mathsf{JNb} \cup \{\mathsf{ebs}\}$*, and a sequent particle $\underline{\kappa}$ in $\mathscr{D}$, there is a derivation in the same system with the same premise and conclusion which is weakly linear in $\underline{\kappa}$.*

Proof. We proceed by permutations of the contractions that are non-weak for $\underline{\kappa}$ in the derivation $\mathscr{D}$, by induction on the number of such instances. If there is no such instance, the result is trivial. Otherwise, we consider the bottommost contraction r non-weak for $\underline{\kappa}$ and use another induction, on the height of the derivation between $\underline{\kappa}$ and r, that we call $\mathscr{D}_1$, to show that we can permute r down to $\underline{\kappa}$. For that, we use a case analysis on the instance r_1 below r, at the top of the $\mathscr{D}_1$ derivation, where $\underline{\iota}$ is the whole sequent particle duplicated by r:

1. If $\underline{\iota}$ is not a proper ancestor of a particle in the conclusion of r_1, this is a case of trivial permutation, and we can use the induction hypothesis since the height of $\mathscr{D}_1$ has decreased.

2. If all active particles in the premise of r_1 are also in $\underline{\iota}$ then we can permute r down. We proceed using the transformation shown below, and we can apply the induction hypothesis since the height of $\mathscr{D}_1$ has decreased:

$$\mathsf{c}\,\dfrac{\mathsf{r}_2\,\dfrac{\xi\{\Gamma, \iota, \iota \vdash A\}}{\xi\{\Gamma, \iota \vdash A\}}}{\xi\{\Gamma, \delta \vdash A\}} \quad\longrightarrow\quad \mathsf{r}_2\,\dfrac{\mathsf{c}\,\dfrac{\xi\{\Gamma, \iota, \iota \vdash A\}}{\xi\{\Gamma, \delta, \iota \vdash A\}}}{\xi\{\Gamma, \delta, \delta \vdash A\}}$$

3. If r_2 is a switch moving some $\underline{\delta}$ inside $\underline{\iota}$, we duplicate it, and the derivation:

$$\mathsf{c}\,\dfrac{\mathsf{s}\,\dfrac{\xi\{\Gamma, [\Delta, [\Psi, \delta \vdash A] \vdash B], [\Delta, [\Psi, \delta \vdash A] \vdash B] \vdash C\}}{\xi\{\Gamma, [\Delta, [\Psi, \delta \vdash A] \vdash B] \vdash C\}}}{\xi\{\Gamma, \delta, [\Delta, [\Psi \vdash A] \vdash B] \vdash C\}}$$

   is then replaced by a new derivation using two contractions, where only one of the contractions is non-weak for $\underline{\kappa}$ — the bottommost one — and we can use the induction hypothesis since the height of $\mathscr{D}_1$ has decreased.

In the case where the contraction non-weak for $\underline{\kappa}$ is not the one used on $\underline{\delta}$, this derivation is the following:

$$\mathsf{c}\,\dfrac{\mathsf{c}\,\dfrac{\mathsf{s}\,\dfrac{\mathsf{s}\,\dfrac{\xi\{\Gamma,[\Delta,[\Psi,\delta\vdash A]\vdash B],[\Delta,[\Psi,\delta\vdash A]\vdash B]\vdash C\}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A]\vdash B],[\Delta,[\Psi,\delta\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,\delta,[\Delta,[\Psi\vdash A]\vdash B],[\Delta,[\Psi\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A]\vdash B],[\Delta,[\Psi\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A]\vdash B]\vdash C\}}$$

At some point, we reach the case where moving r down makes it weak for $\underline{\kappa}$, and we can use the main induction hypothesis, since there is a non-weak instance less.

Note that this is trivially extended to deal with rules with built-in switch in a way similar to case 3. Indeed, consider the rules esa and asa, where both the switch and the contraction are built-in, as an example:

$$\mathsf{esa}\,\dfrac{\Gamma,[[\Gamma\vdash A]\vdash A]\vdash B}{\Gamma\vdash B}\qquad\qquad \mathsf{asa}\,\dfrac{\Gamma,[\Delta,[\Gamma\vdash A]\vdash B]\vdash C}{\Gamma,[\Delta\vdash A\to B]\vdash C}$$

They can be permuted down in a proof. In particular, the application rule asa can be permuted down to the point of the proof where the involved $A\to B$ formula was introduced — this can be the conclusion of the whole proof, or the cut instance which introduced this formula in the antecedent of a sequent. The cut rule esa can also be permuted down, since apart from the antecedent duplication and splitting, it only introduces a sequent in the antecedent of another one. Thus, it can be moved down to introduce the sequent as soon as possible, in terms of proof construction. From this we conclude that any derivation can be made weakly linear. □

Observation 2.9. *For any derivation $\mathscr{D}$, the result $\mathscr{D}'$ of applying Lemma 2.8 to some sequent in $\mathscr{D}$ is such that $\mathscr{H}(\mathscr{D}')\geq\mathscr{H}(\mathscr{D})$.*

The process of making a derivation weakly linear in any sequent particle can be generalised to rule instances, which can be made linear, by permuting contractions down under this instance, and by extension some derivation $\mathscr{D}$ is said to be *weakly linear* if it is linear in all its rule instances — equivalently, if it is weakly linear in all its sequent particles.

Proposition 2.10. *For any derivation $\mathscr{D}$ in the* $\mathsf{JN}\cup\{\mathsf{e}\}$, $\mathsf{JNs}\cup\{\mathsf{es}\}$, $\mathsf{JNa}\cup\{\mathsf{esa}\}$ *or* $\mathsf{JNb}\cup\{\mathsf{ebs}\}$ *proof system there is a weakly linear derivation in the same system, with the same premise and conclusion as $\mathscr{D}$.*

Proof. By induction on the multiset of the multiplicities of all the sequent particles in which the given proof $\mathscr{D}$ is not weakly linear, under multiset ordering. At each step, we pick such a particle $\underline{\kappa}$ and use Lemma 2.8 to make $\mathscr{D}$ weakly linear in $\underline{\kappa}$. It cannot make any rule instance in $\mathscr{D}$ non-weak for a sequent particle if it was not already, since permuting down a contraction preserves weakness: if r is weak for some $\underline{\kappa}$, permuting a contraction under r cannot make it non-weak for $\underline{\kappa}$. Finally, whenever a particle is duplicated in the process, the multiplicity of its copies must be smaller than the original multiplicity, by definition. □

Remark 2.11. *The system* $\mathsf{JNr} \cup \{\mathsf{ers}\}$ *is so restricted that its derivations are almost weakly linear. Consider for example the following derivation:*

$$\mathsf{cr}\,\dfrac{\dfrac{\Gamma, [\vdash A], [\vdash A] \vdash C}{\Gamma, [\vdash A] \vdash C}}{\mathsf{ers}\,\dfrac{\Gamma, [[B \vdash A] \vdash A] \vdash C}{\Gamma, B \vdash C}}$$

The contraction instance here is non-weak for $[B \vdash A] \vdash A$ *but it cannot be permuted down to make it weak — it is located at the bottommost possible position.*

2.3 Merging Proofs

Our cut elimination procedure will make crucial use of a complex technical lemma, typical of deep inference systems, which allows to *merge* a subproof in the hole of a context, inside another proof. It should normally be used to plug a subproof inside a context with only one hole, but in the general case we need to make a stronger statement, because the number of holes can increase during induction — so that we have to use contexts with several holes, as described in (5).

After we give a proof of this lemma, which induces a complex transformation on derivations, we will discuss different situations where this generic transformation is simplified because of the shape of the given derivation — in particular, the linearity of the initial proof in several sequent particles involved is important.

Lemma 2.12 (Merging)**.** *For any proof* $\mathscr{D}$ *of* $\xi\{\Gamma, [\Delta, \delta, \zeta\{\Psi_i \vdash A_i\}^+ \vdash B] \vdash C\}$ *in the* $\mathsf{JNx} \cup \{\mathsf{ex}\}$ *system, there is also a proof of* $\xi\{\Gamma, [\Delta, \zeta\{\Psi_i, \delta \vdash A_i\}^+ \vdash B] \vdash C\}$.

Proof. We consider a particle $\underline{\kappa}$ corresponding to the sequent $\Delta, \delta, \zeta\{\Psi_i \vdash A_i\}^+ \vdash B$ in the conclusion of $\mathscr{D}$, and we proceed by induction on the pair $(\mathscr{M}_{\mathscr{D}}(\underline{\kappa}), \mathscr{H}(\mathscr{D}))$, under lexicographic order. At each step we use a case analysis on the bottommost instance r in $\mathscr{D}$, to rewrite this instance into a derivation of the desired conclusion.

1. If r affects only δ, we use the induction hypothesis on the proof $\mathscr{D}_1$ above r and use the result to build a new proof, with at the bottom the instance r used on δ once inside each copy of the sequent in ζ:

$$\mathsf{r}\,\dfrac{\xi\{\Gamma, [\Delta, \delta', \zeta\{\Psi_i \vdash A_i\}^+ \vdash B] \vdash C\}}{\xi\{\Gamma, [\Delta, \delta, \zeta\{\Psi_i \vdash A_i\}^+ \vdash B] \vdash C\}} \quad\longrightarrow\quad \mathsf{r}^*\,\dfrac{\xi\{\Gamma, [\Delta, \zeta\{\Psi_i, \delta' \vdash A_i\}^+ \vdash B] \vdash C\}}{\xi\{\Gamma, [\Delta, \zeta\{\Psi_i, \delta \vdash A_i\}^+ \vdash B] \vdash C\}}$$

   In the special case where r is an identity instance and deletes all of δ, there is no need to use the induction hypothesis, we are done — since we start with a proof, which is a derivation with no premise, this case must eventually occur.

2. If r does not affect δ, we can proceed by induction hypothesis — this includes the case of a contraction or weakening inside ζ affecting some of the copies of $\Psi_i \vdash A_i$, and this is the reason for the general statement of the theorem to involve an arbitrary number of holes in ζ.

3. If r is a switch moving another sequent υ inside δ, we can apply the induction hypothesis to the proof $\mathscr{D}_1$ above r and use several contractions and switches at the bottom of the resulting proof:

$$\mathsf{s}\,\frac{\xi\{\Gamma,[\Delta,\delta',\zeta\{\Psi_i\vdash A_i\}^+\vdash B]\vdash C\}}{\xi\{\Gamma,\upsilon,[\Delta,\delta,\zeta\{\Psi_i\vdash A_i\}^+\vdash B]\vdash C\}}\quad\longrightarrow\quad\begin{array}{l}\mathsf{s}^*\\ \mathsf{c}^*\end{array}\frac{\xi\{\Gamma,[\Delta,\zeta\{\Psi_i,\delta'\vdash A_i\}^+\vdash B]\vdash C\}}{\xi\{\Gamma,\upsilon,\cdots,\upsilon,[\Delta,\zeta\{\Psi_i,\delta\vdash A_i\}^+\vdash B]\vdash C\}}}{\xi\{\Gamma,\upsilon,[\Delta,\zeta\{\Psi_i,\delta\vdash A_i\}^+\vdash B]\vdash C\}}$$

4. If r is a weakening which deletes the sequent $\Delta,\delta,\zeta\{\Psi_i\vdash A_i\}^+\vdash B$, then we can simply rewrite the conclusive sequent, and there is no need to apply the induction hypothesis, we are done.

5. If r is a contraction which duplicates the sequent $\Delta,\delta,\zeta\{\Psi_i\vdash A_i\}^+\vdash B$, we can rewrite the conclusive sequent and the premise, and apply the induction hypothesis twice — this is possible, since the multiplicity of $\underline{\kappa}$ has decreased.

The cases of variant rules can be treated the same way. Note that in systems where the switch is built-in, there is no need for introducing instances as in case 3, since no material can be moved to a sequent that already exists. □

Observation 2.13. *For any given proof $\mathscr{D}$, the result $\mathscr{D}'$ of applying Lemma 2.12 to $\mathscr{D}$ is in general such that $\mathscr{H}(\mathscr{D}')\geq\mathscr{H}(\mathscr{D})$.*

The problem with Lemma 2.12 is that it might duplicate parts of the given proof and introduce new rule instances, in particular new contractions, at various places within the proof, which is a complicated non-local rewriting of the proof. However, it behaves well on proofs that respect some restrictions — we use the names from the statement of the lemma:

1. If the context $\zeta\{\ \}^+$ has only one hole filled with $\Psi\vdash A$, and if the proof $\mathscr{D}$ is linear in this sequent particle, then cases 4 and 5 are never used and case 2 never involves duplication of the hole in ζ, so that no contraction will need to be introduced in case 3.

2. If no ancestor of $\underline{\delta}$ is modified by a rule instance located above an instance affecting $\zeta\{\Psi_i\vdash A_i\}^+$, and if $\zeta\{\ \}^+$ has only one hole, then no rule instance needs to be duplicated, and no contraction needs to be introduced.

We can therefore improve the situation here by using Lemma 2.12 only on proofs of a particular shape, for example by applying weak linearisation, so as to fit the first criterion described above. We could also have a situation where the subproof to be merged into a context is completed before the context starts being modified, which validates the second criterion. To enforce such a situation, we can transform the given proof, before applying Lemma 2.12, as follows:

$$\begin{array}{c}\mathscr{D}\Vdots\\ \xi\{\Gamma,[\Delta,\delta,\zeta\{\Psi_i\vdash A\}^+\vdash B]\vdash C\}\end{array}\quad\longrightarrow\quad\begin{array}{c}\mathscr{D}_2\Vdots\\ \eta\{\Sigma_j,[\Xi_j,\zeta\{\Psi_i\vdash A_i\}^+\vdash D_j]\vdash E_j\}^+\\ \mathscr{D}_1\Vdots\\ \xi\{\Gamma,[\Delta,\delta,\zeta\{\Psi_i\vdash A_i\}^+\vdash B]\vdash C\}\end{array}$$

where there is no ancestor of δ in $\mathscr{D}_2$. This process allows to *extract* the proof of the positive sequent δ from $\mathscr{D}$, to separe it from instances affecting $\zeta\{\ \}^+$.

The idea is that, when applying Lemma 2.12 to a proof obtained this way, $\underline{\delta}$ is simply moved inside the context $\zeta\{\ \}^+$, but there is no need to handle rule instances modifying this context — and contractions cannot duplicate $\zeta\{\ \}^+$ independently from $\underline{\delta}$ or its ancestors, so that there is no need to introduce new contractions. In the case where there is only one hole in ζ, the merging procedure is straightforward and does not require any duplication.

Lemma 2.14. *Any proof $\mathscr{D}$ of $\xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\}$ in the $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$ system can be decomposed into a derivation $\mathscr{D}_1$ and a proof $\mathscr{D}_2$, such that in $\mathscr{D}_1$ no ancestor of the particle $\underline{\kappa}$ is modified, and in $\mathscr{D}_2$ no ancestor of the particle $\underline{\delta}$ appears.*

Proof. The idea is to push the instances affecting $\underline{\kappa}$ and its ancestors upwards until they are all located above the topmost instance affecting an ancestor of $\underline{\delta}$. For that, we consider that $\mathscr{D}$ is decomposed into two derivations $\mathscr{D}_1$ and $\mathscr{D}_3$, and a proof $\mathscr{D}_4$, as illustrated below:

$$\begin{array}{c} \mathscr{D}_4 \Vdots \\ \eta\{\upsilon_j\}^+ \\ \mathscr{D}_3 \Vdots \\ \zeta\{\kappa_i\}^+ \\ \mathscr{D}_1 \Vdots \\ \xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\} \end{array}$$

- In the derivation $\mathscr{D}_1$, the particle $\underline{\kappa}$ and its ancestors are not modified, they have the same basis, but they can be duplicated or erased — the premise of $\mathscr{D}_1$ is a sequent $\zeta\{\kappa_i\}^+$, where the $\underline{\kappa_i}$ are all the ancestors of $\underline{\kappa}$ at this point.
- In the derivation $\mathscr{D}_3$, ancestors of $\underline{\kappa}$ can be modified, but no ancestor of $\underline{\delta}$ can be active in a rule instance, so that in particular, ancestors of $\underline{\kappa}$ and $\underline{\delta}$ cannot be duplicated or erased.
- The proof $\mathscr{D}_4$ is arbitrary and contains rule instances that may or may not modify, duplicate or erase any ancestor of $\underline{\kappa}$ or $\underline{\delta}$.

We proceed by induction on $\mathscr{H}(\mathscr{D}_4)$, and the base case happens when $\mathscr{D}_4$ is empty, and thus the decomposition into $\mathscr{D}_1$ and $\mathscr{D}_3$ is the expected result — in this case, $\mathscr{D}_3$ is the proof $\mathscr{D}_2$ of the statement — or when $\mathscr{D}_4$ contains no ancestor of δ, so that $\mathscr{D}_3$ cannot contain ancestors of $\underline{\delta}$ as well, and $\mathscr{D}_2$ is the composition of $\mathscr{D}_3$ and $\mathscr{D}_4$. Moreover, any proof of $\xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\}$ can be decomposed into derivations $\mathscr{D}_1$, $\mathscr{D}_3$ and $\mathscr{D}_4$ as described, at least with $\mathscr{D}_1$ and $\mathscr{D}_3$ empty. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}_4$:

1. If r is an instance where no ancestor of $\underline{\kappa}$ is active, we trivially permute it down under $\mathscr{D}_3$ to merge it into $\mathscr{D}_1$, and go on by induction hypothesis:

$$\begin{array}{c} \mathsf{r}\dfrac{\eta'\{\upsilon_j\}^+}{\eta\{\upsilon_j\}^+} \\ \mathscr{D}_3 \Vdots \\ \zeta\{\kappa_i\}^+ \\ \mathscr{D}_1 \Vdots \\ \xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\} \end{array} \quad \longrightarrow \quad \begin{array}{c} \eta'\{\upsilon_j\}^+ \\ \mathscr{D}_3^* \Vdots \\ \mathsf{r}\dfrac{\zeta'\{\kappa_i\}^+}{\zeta\{\kappa_i\}^+} \\ \mathscr{D}_1 \Vdots \\ \xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\} \end{array}$$

Note that in the case where r is some identity erasing the topmost ancestor of the $\underline{\delta}$ occurrence, we have reached a base case, and since we are working on a proof, such a situation must be reached.

2. If r is a rule instance modifying the structure of an ancestor $\underline{\upsilon_i}$ of $\underline{\kappa}$, we merge r inside $\mathscr{D}_3$ and go on by induction hypothesis — note that such an instance cannot affect an ancestor of $\underline{\delta}$, since $\underline{\kappa}$ and $\underline{\delta}$ are in positive position.

3. If r is a contraction which duplicates an ancestor $\underline{\upsilon_j}$ of $\underline{\kappa}$ into new ancestors $\underline{\upsilon_k}$ and $\underline{\upsilon_l}$, it can also duplicate an ancestor of $\underline{\delta}$, and we have to permute r down under $\mathscr{D}_3$ to merge it into $\mathscr{D}_1$. Since the derivation $\mathscr{D}_3$ does not contain any contraction duplicating ancestors of $\underline{\kappa}$, we can permute r down as done below, using Lemma 2.8 and go on by induction hypothesis:

$$\mathsf{r}\,\dfrac{\dfrac{\eta'\{\Sigma, \theta\{\upsilon_k\}, \theta\{\upsilon_l\} \vdash C\}^+}{\eta'\{\Sigma, \theta\{\upsilon_j\} \vdash C\}^+}\mathscr{D}_3\|}{\zeta\{\kappa_i\}^+}\mathscr{D}_1\|}{\xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\}} \quad\longrightarrow\quad \mathsf{r}\,\dfrac{\dfrac{\eta\{\upsilon_m\}^+}{\mathscr{D}'_3\|}\zeta'\{\Psi, \theta'\{\kappa_k\}, \theta'\{\kappa_l\} \vdash D\}^+}{\zeta'\{\Psi, \theta'\{\kappa_i\} \vdash D\}^+}\mathscr{D}_1\|}{\xi\{\Gamma, [\Delta, \delta, \kappa \vdash A] \vdash B\}}$$

4. If r is a weakening erasing an ancestor $\underline{\upsilon_j}$ of $\underline{\kappa}$, it can also erase an ancestor of $\underline{\delta}$ and we have to permute r down under $\mathscr{D}_3$ to merge it with $\mathscr{D}_1$. This is done the same way as in the previous case, using weak linearisation.

The cases of variant rules can all be treated the same way. Notice in particular that the restricted contraction of the $\mathsf{JNr} \cup \{\mathsf{ers}\}$ system can never match the situation of case 3, and restricted weakening can never match the situation of case 4. □

Observation 2.15. *For any given proof $\mathscr{D}$, the result $\mathscr{D}'$ of applying Lemma 2.14 to $\mathscr{D}$ is in general such that $\mathscr{H}(\mathscr{D}') \geq \mathscr{H}(\mathscr{D})$.*

2.4 Eliminating Cuts

We can now present the cut elimination procedure and its variations, for any proof system $\mathsf{JNx} \cup \{\mathsf{ex}\}$ in the family of all variants of $\mathsf{JN} \cup \{\mathsf{e}\}$. We will give only one proof of the theorem stating the admissibility of the cut rule, providing a procedure to eliminate cuts in the JN system, but this is easily adapted to all variant systems that we defined in the previous section. In any case, the basic idea is to push cut instances up in the proof until they meet other rule instances to interact with, thus disappearing or decomposing in smaller cuts, as in the sequent calculus [Gal93].

The first step here is to define the measure that will be used for the induction when showing that all cuts can be moved upwards to be eliminated. In addition to the general definitions already used, we need a specific measure for cut instances, this is sometimes called the *logical complexity* of the cut.

Definition 2.16. *In a derivation $\mathscr{D}$ in* $\mathsf{JNx} \cup \{\mathsf{ex}\}$*, the* size *of a cut instance r with a principal sequent δ, denoted by $|\mathsf{r}|$, is the number of symbols used in δ.*

Then, we need to manipulate the shape of the derivation above a cut instance, and thus definitions for the different subderivations we will be looking at.

Definition 2.17. *In any derivation $\mathscr{D}$ in* $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$ *if a cut instance* r *has a principal sequent of the shape $\Delta, A \to B \vdash A \to B$, a rule instance is said to be* matching *the cut* r *if and only if it affects an ancestor of one of these particles with basis $A \to B$.*

In particular, the left and right implication instances decomposing ancestors of the particles of $A \to B$ are matching such a cut, as an identity on these ancestors, or a structural rule applied on ancestor of the whole sequent introduced by the cut.

When moving a cut instance upwards in a proof, some rule instances might be encountered that cannot be permuted down under the cut. This happens when a matching left or right implication instance decomposes the cut formula, and some rule instances apply on pieces of the cut formula, but also when a switch moves material to the sequent introduced by the cut. To handle this, we define synthetic cut rules, embedding the cut and all rule instances that should be pushed together with the cut. In the first case, some switch instances are blocked by a cut, so that we have to use a cut with built-in switch, as follows:

$$\mathsf{es}\,\dfrac{\xi\{\Gamma, [[\Delta_1 \vdash A] \vdash A] \vdash B\}}{\xi\{\Gamma, \Delta_1 \vdash B\}} \quad = \quad \mathsf{e}\,\dfrac{\xi\{\Gamma, [[\Delta_1 \vdash A] \vdash A] \vdash B\}}{\xi\{\Gamma, \Delta_1, [A \vdash A] \vdash B\}}\ \mathscr{D}_1\| \qquad (6)$$

In a second possible situation, a cut is composed with a left implication instance and a derivation $\mathscr{D}_1$ made of some rule instances which cannot be permuted down because of the left implication instance, or because they are switches, as follows:

$$\mathsf{ea}\,\dfrac{\xi\{\Gamma, [[\Delta_1 \vdash C \to D], \Psi \vdash E] \vdash B\}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}} = \mathsf{a}\,\dfrac{\mathsf{e}\,\dfrac{\xi\{\Gamma, [[\Delta_1 \vdash C \to D], \Psi \vdash E] \vdash B\}}{\xi\{\Gamma, \Delta_1, \Delta_2, [C \to D, C \vdash D] \vdash B\}}\ \mathscr{D}_1\|}{\xi\{\Gamma, \Delta_1, \Delta_2, [C \to D \vdash C \to D] \vdash B\}} \qquad (7)$$

Finally, in the third possible situation, a cut is composed with a right implication instance and a derivation $\mathscr{D}_1$ made of instances that cannot be permuted down because of this right implication instance, or because they are switches, as follows:

$$\mathsf{eg}\,\dfrac{\xi\{\Gamma, [[\Delta_1, \Psi \vdash E] \vdash C \to D] \vdash B\}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}} = \mathsf{g}\,\dfrac{\mathsf{e}\,\dfrac{\xi\{\Gamma, [[\Delta_1, \Psi \vdash E] \vdash C \to D] \vdash B\}}{\xi\{\Gamma, \Delta_1, \Delta_2, [[C \vdash D] \vdash C \to D] \vdash B\}}\ \mathscr{D}_1\|}{\xi\{\Gamma, \Delta_1, \Delta_2, [C \to D \vdash C \to D] \vdash B\}} \qquad (8)$$

Any cut instance r in a given proof $\mathscr{D}$ matches exactly one of these configurations, and they will form the basis of the case analysis performed during cut elimination, which will be structured around the interaction between the cut, the derivation $\mathscr{D}_1$ embedded with it, and the rule instance directly above the body.

We will treat directly the elimination of all three cut forms in our proof, which entails the elimination of the basic cut rule e, since it is just a particular case of es. Notice that all definitions on cuts can be extended trivially to the synthetic rules ea and eg, by considering the standard cut embedded into it — in particular, their principal sequent is the principal sequent of the basic cut.

Definition 2.18. *In a derivation $\mathscr{D}$ in $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$, the* body *of a cut instance r is the derivation $\mathscr{D}_1$ as indicated in* (6)*,* (7) *and* (8)*, and the* target *of the instance r is the rest of the derivation, above the synthetic cut.*

We now define the measure used on cut instances, built to decrease during cut elimination, either because the size or multiplicity of the principal sequent of one cut decreases, or because some unrelated rule instance is moved below a cut.

Definition 2.19. *In a derivation $\mathscr{D}$ in $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$, the* rank *of a cut instance r, which is denoted by $\mathscr{R}_{\mathscr{D}}(\mathsf{r})$, is the pair $(|\mathsf{r}|, \mathscr{M}_{\mathscr{D}}(\mathsf{r}))$ under lexicographic ordering.*

Our proof of cut elimination is based on a set of rewritings of the given proof, most of them being local to a cut, its body, and the bottommost rule instance above. To avoid dealing with complex relationships between different cuts, we will impose a strategy on the elimination of cuts, so that the cut involved in the rewriting will always be the topmost one in a proof. Thus, we need no definition of a rank for a proof, since we will consider proofs with exactly one cut, as bottommost instance.

Before we can start with the proof of the main theorem, we need to prove a few technical lemmas. The first one is standard, and it corresponds to the well-known invertibility result of the right implication rule in the sequent calculus. The second one is similar[3] but applies to implication in negative position, and then the last two are almost trivial observations, due to the features of deep inference, but we make them explicit to emphasize their simplicity in this setting.

Lemma 2.20. *For any proof $\mathscr{D}$ of a sequent $\xi\{\Gamma \vdash A \to B\}$ in the $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$ system there is also a proof $\mathscr{D}'$ of $\xi\{\Gamma, A \vdash B\}$.*

Proof. By structural induction on $\mathscr{D}$, using a case analysis on the bottommost rule instance r in $\mathscr{D}$. The base case happens when r is a right implication instance which decomposes the implication $A \to B$, and we remove this instance and use the proof above. In the general case, we can rewrite any rule instance by replacing a sequent occurrence $\Psi \vdash A \to B$ with $\Psi, A \vdash B$, and use the induction hypothesis. □

Lemma 2.21. *For any proof $\mathscr{D}$ of a sequent $\xi\{\Gamma, [\Delta \vdash A \to B] \vdash C\}$ in the $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$ system there is a proof $\mathscr{D}'$ of $\xi\{\Gamma, [\Delta, A \vdash B] \vdash C\}$.*

Proof. We proceed the same way as for Lemma 2.20, except in the base case, where the bottommost rule instance in $\mathscr{D}$ should be a left application instance. □

Remark 2.22. *Using these two lemmas, it is always possible to move the left and right implication instances down, so that any given cut instance matches exactly one of the configuration depicted in* (6)*,* (7) *and* (8)*.*

[3] A left implication rule instance cannot be moved downwards in all systems, but the particular shape of this rule in a system where no switch is built-in allows to move it freely.

Lemma 2.23. *For a derivation $\mathscr{D}$ from $\zeta\{\delta\}$ to $\xi\{\delta\}$ in $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$, if $\mathscr{D}$ is linear in the bottommost $\underline{\delta}$, there is a derivation $\mathscr{D}'$ from $\zeta\{\ \}$ to $\xi\{\ \}$.*

Proof. By structural induction on $\mathscr{D}$, using a case analysis on the bottommost rule instance r in $\mathscr{D}$. In the base case, $\mathscr{D}$ is a sequent, and we rewrite it to remove δ. In the general case, r cannot modify $\underline{\delta}$, since a proper ancestor of $\underline{\delta}$ appears in the premise of $\mathscr{D}$, and it cannot be a structural rule instance since $\mathscr{D}$ is linear in this particle. If it does not affect $\underline{\delta}$, we rewrite it to remove $\underline{\delta}$ and go on by induction hypothesis. Otherwise, it must be a switch moving $\underline{\delta}$ inside another sequent κ, and we can remove this instance and go on by induction hypothesis. □

Lemma 2.24. *For a derivation $\mathscr{D}$ from $\zeta\{\Gamma, \delta \vdash A\}$ to $\xi\{\Delta, \delta \vdash B\}$ in $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$, if $\mathscr{D}$ is linear in the bottommost $\underline{\delta}$, there is a derivation $\mathscr{D}'$ from $\zeta\{\Gamma \vdash A\}$ to $\xi\{\Delta \vdash B\}$.*

Proof. The proof is exactly the same as the proof of Lemma 2.23, since it did not rely on the fact that $\underline{\delta}$ was in positive position. □

Observation 2.25. *For any derivation or proof $\mathscr{D}$, the result $\mathscr{D}'$ of applying Lemma 2.20, Lemma 2.21, Lemma 2.23 or Lemma 2.24 to $\mathscr{D}$ is such that $\mathscr{H}(\mathscr{D}') \leq \mathscr{H}(\mathscr{D})$.*

We can now come to the cut elimination result itself. It is obtained by induction, using a case analysis. Each case of this analysis corresponds to a rewriting step.

Theorem 2.26 (Cut elimination)**.** *A proof $\mathscr{D}$ of a nested sequent $\Gamma \vdash B$ in $\mathsf{JN}\mathsf{x} \cup \{\mathsf{ex}\}$ can be transformed into a cut-free proof $\mathscr{D}'$ of $\Gamma \vdash B$ in $\mathsf{JN}\mathsf{x}$.*

Proof. If there is no cut instance in the given proof $\mathscr{D}$, we are done. In the general case, we proceed by induction on the number of cut instances in $\mathscr{D}$, and consider the topmost cut instance r in $\mathscr{D}$, with a target that we name $\mathscr{D}_2$. Then, we proceed by induction on the pair $(\mathscr{R}_{\mathscr{D}}(\mathsf{r}), \mathscr{H}(\mathscr{D}_2))$, to show that this cut can be eliminated. For that, we use a case analysis on the bottommost instance r_1 in $\mathscr{D}_2$:

$$\mathsf{r}\,\dfrac{\mathsf{r}_1\,\dfrac{\begin{array}{c}\mathscr{D}_2\Vdots\\ \zeta\{\upsilon\}\end{array}}{\xi\{\Gamma, \kappa \vdash B\}}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}}$$

1. If r_1 is not in the scope of r, we can permute it down and proceed by induction hypothesis, because the height of $\mathscr{D}_2$ has decreased.
2. If r_1 is a switch moving material into the sequent κ introduced by the cut, or affects only the inside $\Psi \vdash E$ of κ, we can assimilate it into the cut $\mathsf{es}/\mathsf{ea}/\mathsf{eg}$ instance, since r_1 can be made part of the body $\mathscr{D}_1$ of the cut, as shown above in (6), (7) and (8), and then we can use the induction hypothesis, since the height of $\mathscr{D}_2$ has decreased.

Thus, we only need to consider the cases in which r_1 affects the cut formula A or $C \to D$ that was introduced, and is blocked above the cut. This leaves the cases of an identity matching the cut in any way, a matching weakening or contraction, and the main case of both matching left and right implication instances.

3. If r_1 is an identity instance matching r, it can interact with r and disappear, and we can apply the main induction hypothesis, since a cut was eliminated by the following transformation, in the case of a simple es cut:

$$\mathsf{i}\,\dfrac{\mathsf{es}\,\dfrac{\xi\{\Gamma, A \vdash B\}}{\xi\{\Gamma, [[A \vdash A] \vdash A] \vdash B\}}}{\xi\{\Gamma, A \vdash B\}} \quad\longrightarrow\quad \xi\{\Gamma, A \vdash B\}$$

In the case of a cut which is an ea instance, the body $\mathscr{D}_1$ of the cut does not disappear, but we use the following transformation, where $\mathscr{D}_1'$ is obtained by applying Lemma 2.23 and then Lemma 2.24 to $\mathscr{D}_1$, which is possible because it is linear in both $C \to D$ particles in its conclusion:

$$\mathsf{i}\,\dfrac{\mathsf{ea}\,\dfrac{\xi\{\Gamma, [\Psi \vdash E] \vdash B\}}{\xi\{\Gamma, [[C \to D \vdash C \to D], \Psi \vdash E] \vdash B\}}}{\xi\{\Gamma, C \to D, \Delta_2 \vdash B\}} \quad\longrightarrow\quad \mathsf{a}\,\dfrac{\begin{array}{c}\xi\{\Gamma, [\Psi \vdash E] \vdash B\}\\ \mathscr{D}_1'\|\\ \xi\{\Gamma, [C \vdash D], \Delta_2 \vdash B\}\end{array}}{\xi\{\Gamma, C \to D, \Delta_2 \vdash B\}}$$

and we can use the main induction hypothesis on the resulting proof, since a cut instance was erased.

Finally, in the case where the cut is an eg instance, it can happen that its body $\mathscr{D}_1$ is actually a proof of the sequent $\Delta_1, \Delta_2, C \vdash D$, so that $\mathscr{D}_1$ is kept and we use the following transformation, where the proof $\mathscr{D}_1'$ is again obtained by applying Lemma 2.23 and Lemma 2.24 on $\mathscr{D}_1$:

$$\mathsf{i}\,\dfrac{\mathsf{eg}\,\dfrac{\xi\{\,\}}{\xi\{C \to D \vdash C \to D\}}}{\xi\{\Delta_1, \Delta_2 \vdash C \to D\}} \quad\longrightarrow\quad \mathsf{g}\,\dfrac{\begin{array}{c}\xi\{\,\}\\ \mathscr{D}_1'\|\\ \xi\{\Delta_1, \Delta_2, C \vdash D\}\end{array}}{\xi\{\Delta_1, \Delta_2 \vdash C \to D\}}$$

Again, in this situation, we can use the main induction hypothesis, because one cut instance was erased from the proof.

4. If r_1 is a weakening erasing the principal sequent of r, we replace the whole body $\mathscr{D}_1$ by weakenings, as shown below in the case where r_1 erases only the sequent κ and not a broader sequent, and then go on by induction hypothesis, since one cut was erased by this transformation:

$$\mathsf{es/ea/eg}\,\dfrac{\mathsf{w}\,\dfrac{\xi\{\Gamma \vdash B\}}{\xi\{\Gamma, \kappa \vdash B\}}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}} \quad\longrightarrow\quad \mathsf{w}^*\,\dfrac{\xi\{\Gamma \vdash B\}}{\xi\{\Gamma, \Delta \vdash B\}}$$

The transformation is similar in the cases where some sequent broader than κ, which contains κ, is erased by this weakening instance, but then no new weakening needs to be introduced.

5. If r_1 is a contraction duplicating the principal sequent of r, we duplicate the whole body $\mathscr{D}_1$ and compose the two copies, one above the other, as shown below in the case where r_1 duplicates only the sequent κ and not a broader sequent in which κ is contained:

$$\mathsf{es/ea/eg}\,\dfrac{\mathsf{c}\,\dfrac{\xi\{\Gamma,\kappa,\kappa\vdash B\}}{\xi\{\Gamma,\kappa\vdash B\}}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}} \quad\longrightarrow\quad \mathsf{es/ea/eg}\,\dfrac{\mathsf{es/ea/eg}\,\dfrac{\xi\{\Gamma,\kappa,\kappa\vdash B\}}{\xi\{\Gamma,\kappa,\Delta_1,\Delta_2\vdash B\}}}{\mathsf{c}^*\,\dfrac{\xi\{\Gamma,\Delta_1,\Delta_2,\Delta_1,\Delta_2\vdash B\}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}}$$

We can use the induction hypothesis on the topmost copy of the cut, because its multiplicity is smaller than the multiplicity of the original one. Then, it is also possible to apply the induction hypothesis on the resulting proof glued above the bottommost copy, although the multiplicity of this cut might have increased after the elimination of the topmost one — through the use of the merging lemma, as shown in the following cases. Indeed, each contraction affecting this cut introduced by merging was introduced for one contraction affecting the topmost copy, so that the multiplicity of the bottommost copy is at most one less than the original multiplicity. Finally, we can use the main induction hypothesis, since one cut was eliminated.

The transformation is similar in the cases where a sequent broader than κ, which contains κ, is duplicated by the contraction, but no new contraction needs to be introduced in this case.

6. If r_1 is a left implication instance and r is an instance of eg, then we have the situation described below:

$$\mathsf{eg}\,\dfrac{\mathsf{a}\,\dfrac{\xi\{\Gamma,[[\Delta_1,\Psi\vdash E],C\vdash D]\vdash B\}}{\xi\{\Gamma,[[\Delta_1,\Psi\vdash E]\vdash C\to D]\vdash B\}}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}$$

and we use Lemma 2.21 on the body $\mathscr{D}_1$ of this cut, to produce a new proof where the basic cut and the left and right implication instances are grouped one above the other, as shown below on the left.

Then, we can rewrite the cut r into two smaller cuts, as shown below on the right, where the proof $\mathscr{D}_3$ above the new cuts is obtained from the body and the target of the original cut.

$$\mathsf{e}\,\dfrac{\mathsf{a}\,\dfrac{\mathsf{g}\,\dfrac{\xi\{\Gamma,\Delta_1,\Delta_2,[[C\vdash D],C\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta_1,\Delta_2,[C\to D,C\vdash D]\vdash B\}}}{\xi\{\Gamma,\Delta_1,\Delta_2,[C\to D\vdash C\to D]\vdash B\}}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}\quad\text{with}\quad \begin{array}{c}\mathscr{D}_2'\Vdash\\ \xi\{\Gamma,[[\Delta_1,\Psi\vdash E],C\vdash D]\vdash B\}\end{array}\quad\text{and}\quad \mathscr{D}_1'\Vdash$$

$$\longrightarrow\quad \mathsf{e}\,\dfrac{\mathsf{e}\,\dfrac{\xi\{\Gamma,\Delta_1,\Delta_2,[[[C\vdash C]\vdash D]\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta_1,\Delta_2,[D\vdash D]\vdash B\}}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}\quad\text{with}\quad \mathscr{D}_3\Vdash$$

However, this rewriting of the cut into two cuts changes the premise of the cut, so that we need to adapt the proof above these new cuts to glue all the parts together. This is done using Lemma 2.12, which applies merging to the proof above the right implication instance, and produces the proof $\mathscr{D}_3$. We can then use the induction hypothesis on the topmost copy of the cut and $\mathscr{D}_3$, since this copy has a smaller size than the original, and we can apply it again on the bottommost copy for the same reason. Finally, we can apply the main induction hypothesis, since one cut was eliminated.

7. If r_1 is a right implication instance and r is an instance of ea, then we are in the situation described below:

$$\dfrac{\dfrac{\xi\{\Gamma, [[\Delta_1, C \vdash D], \Psi \vdash E] \vdash B\}}{\xi\{\Gamma, [[\Delta_1 \vdash C \to D], \Psi \vdash E] \vdash B\}}\,\mathsf{g}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{ea}$$

and we proceed as in the previous case, applying Lemma 2.20 to produce a proof where the cut and the left and right implication instances are grouped, and rewrite the cut r into two smaller cuts, as shown below:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma, [[\Delta_1, C \vdash D], \Psi \vdash E] \vdash B\}}{\xi\{\Gamma, \Delta_1, \Delta_2, [[C \vdash D], C \vdash D] \vdash B\}}\,\mathscr{D}_2'\!\!\parallel\!\!\mathscr{D}_1'}{\xi\{\Gamma, \Delta_1, \Delta_2, [[C \vdash D] \vdash C \to D] \vdash B\}}\,\mathsf{a}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{g}\,\mathsf{e} \quad\longrightarrow\quad \dfrac{\dfrac{\xi\{\Gamma, \Delta_1, \Delta_2, [[[C \vdash C] \vdash D] \vdash D] \vdash B\}}{\xi\{\Gamma, \Delta_1, \Delta_2, [D \vdash D] \vdash B\}}\,\mathscr{D}_3\!\!\parallel}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{e}$$

As before, this rewriting of the cut changes the premise of the cut, so that we need to adapt the proof above the new cuts to glue the parts together. This is done using Lemma 2.12, which applies merging to the proof above the left implication instance, producing $\mathscr{D}_3$. We can use the induction hypothesis on the topmost copy of the cut and $\mathscr{D}_3$, since this copy has a smaller size than the original, and we can apply it again on the bottommost copy for the same reason. Finally, we apply the main induction hypothesis, which is possible because one cut was eliminated.

The cases of all variants of the basic rules can be dealt with the same way. Indeed, systems where the switch rule is built inside the rules of cut and left implication are even simpler to handle because proofs have a more constrained shape — and the synthetic cut es is already in the system. When structural rules are built-in, cases 4 and 5 happen only when a cut encounters an application, and the treatment is the same. Moreover, the merging procedure which is implemented by Lemma 2.12 and crucially used in the principal cases 6 and 7 can be applied on any given proof in any of the variant system we defined. □

Remark 2.27. *One should notice that in this cut elimination procedure, there is only one configuration where a cut instance interacts with a an identity instance, described in case 3. We could define, and prove sound, an extension* is *of the basic identity rule similar to the extension* es *of the basic cut rule, as follows:*

$$\mathsf{is}\ \frac{\Gamma, [\Delta, \Psi \vdash B] \vdash C}{\Gamma, [\Delta, [[\Psi \vdash A] \vdash A] \vdash B] \vdash C}$$

and this would induce a another interaction configuration, where the right-hand side of the principal sequent in the cut instance would thus match the right-hand side of the identity instance:

$$\begin{array}{l}\mathsf{is}\\ \mathsf{es}\end{array}\ \frac{\dfrac{\xi\{\Gamma, [\Delta, [\Psi \vdash A] \vdash B] \vdash C\}}{\xi\{\Gamma, [\Delta, [[[\Psi \vdash A] \vdash A] \vdash A] \vdash B] \vdash C\}}}{\xi\{\Gamma, [\Delta, [\Psi \vdash A] \vdash B] \vdash C\}} \quad \longrightarrow \quad \xi\{\Gamma, [\Delta, [\Psi \vdash A] \vdash B] \vdash C\}$$

This cut elimination procedure can be considered in different situations, each one inducing a variation on the general scheme where cuts are pushed upwards in the proof to interact with identities. This includes situations where the proof is weakly linear in some particles, where the invertibility result of Lemma 2.20 and Lemma 2.21 is used or not, or where merging is performed by Lemma 2.12 together with Lemma 2.14 or with partial weak linearisation. We can give a quick overview of these situations by listing various possibilities:

1. If we use Proposition 2.10 on a proof before we start cut elimination to make it weakly linear, there is no need to handle contractions in case 5, because no such instance can affect the principal sequent of a cut. Moreover, this reduces the use of the merging lemma to the case where no contraction needs to be introduced because the sequent moved around is never duplicated.

2. If we directly use the invertibility result from Lemma 2.20 and Lemma 2.21, we can reduce immediately a cut to the atomic shape, using merging. Then, if there are matching left and right implication instances for all cuts — which means that all identity instances are also atomic, which can be done as shown in Proposition 1.5 — we only need to deal with atomic cuts.

3. If we always use Lemma 2.14 to extract subproofs before using Lemma 2.12, no contraction is introduced during merging, but some might be needed to perform the extraction. To avoid dealing with such contractions, we can use weak linearisation by applying Lemma 2.8 on the principal sequent of the considered cut instance.

4. If we use the weak linearisation transformation, by using Lemma 2.8 on the principal sequent of the considered cut and on the sequent used as a context during merging, then the use of the merging lemma in cases 6 and 7 requires the introduction of contractions, but all these new contractions are located at the bottom of the proof produced.

Although all variations of the proof of Theorem 2.26 provide the same result that all variants of the cut rule are admissible in their respective systems, the details of the cut elimination procedure are crucial when it comes to the exploration the computational contents of proofs. Indeed, different manipulations on proofs are reflected as different rewriting rules on the terms which represent the proof objects.

3 Local Normalisation

Although the procedure we devised to eliminate cuts from a given proof in one of our nested sequents system for intuitionistic logic is quite close to the one used in the standard sequent calculus, it is not based only on local rewritings. Indeed, the main case relies on the use of the merging lemma, which implements a complicated non-local rewriting of the whole proof located above the chosen cut. In order to solve this problem, we would like to find a derivation, or simply an inference rule, that would be composed with the two new cuts and restore the original form of the premise, as done by the rule r below:

$$\mathsf{e}\,\dfrac{\mathsf{a}\,\dfrac{\mathsf{g}\,\dfrac{\xi\{\Gamma,\Delta,[[C\vdash D],C\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta,[[C\vdash D]\vdash C\to D]\vdash B\}}}{\xi\{\Gamma,\Delta,[C\to D\vdash C\to D]\vdash B\}}}{\xi\{\Gamma,\Delta\vdash B\}} \quad\longrightarrow\quad \mathsf{e}\,\dfrac{\mathsf{r}\,\dfrac{\xi\{\Gamma,\Delta,[[C\vdash D],C\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta,[[[C\vdash C]\vdash D]\vdash D]\vdash B\}}}{\mathsf{e}\,\dfrac{\xi\{\Gamma,\Delta,[D\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta\vdash B\}}}$$

Here, the rule r embodies exactly the merging operation stated by Lemma 2.12, moving a positive sequent deeper inside another positive sequent. This means that the merging lemma is actually a proof of admissibility of the rule r, so that we can simply use this rule in one of the $\mathsf{JN}\times\cup\{\mathsf{ex}\}$ proof systems.

More interestingly, this rule r happens to be the *dual* of the switch rule s, which moves a negative sequent deeper inside another negative sequent — it is the dual in the same sense as the cut e rule being dual of the identity i rule, as usual in a deep inference setting [Gug07]. Moreover, if we introduce this rule in the system, we would like to be able to eliminate it, as we eliminate cuts, since its dual is already present in the system. This is also the standard expectation for a deep inference system, where all dual rules are considered as part of what the cut rule embodies in the sequent calculus [Brü06a, Str03b]. As we will see, eliminating this rule requires the use of other dual rules, so that we get closer to the usual design of proof systems in the *calculus of structures*, where the basic system, the *down fragment*, is enriched with the set of corresponding dual rules, the *up fragment*, which are all admissible for the down fragment — the dual of a rule is obtained by reversing premise and conclusion and inverting all polarities, so that positive sequents become negative, and negative ones become positive.

We present now a symmetric variant of the basic JN system, called SJN, and for which the inference rules are given in Figure 3. The first subsystem, where all rules have names of the shape $\mathsf{r}{\downarrow}$, is the *down fragment*, and the second one, where names are of the shape $\mathsf{r}{\uparrow}$, is the *up fragment*. The down fragment corresponds to the cut-free system JN, and the up fragment consists in the duals of all rules in JN.

$$\mathsf{i}{\downarrow}\ \frac{}{A \vdash A} \qquad \mathsf{c}{\downarrow}\ \frac{\Gamma, \delta, \delta \vdash A}{\Gamma, \delta \vdash A} \qquad \mathsf{g}{\downarrow}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \qquad \mathsf{a}{\downarrow}\ \frac{\Gamma, [\Delta, A \vdash B] \vdash C}{\Gamma, [\Delta \vdash A \to B] \vdash C}$$

$$\mathsf{w}{\downarrow}\ \frac{\Gamma \vdash A}{\Gamma, \delta \vdash A} \qquad \mathsf{s}{\downarrow}\ \frac{\Gamma, [\Delta, [\Psi, \delta \vdash A] \vdash B] \vdash C}{\Gamma, \delta, [\Delta, [\Psi \vdash A] \vdash B] \vdash C}$$

$$\mathsf{i}{\uparrow}\ \frac{\Gamma, [A \vdash A] \vdash B}{\Gamma \vdash B} \qquad \mathsf{w}{\uparrow}\ \frac{\Delta, [\Gamma, \delta \vdash A] \vdash B}{\Delta, [\Gamma \vdash A] \vdash B} \qquad \mathsf{s}{\uparrow}\ \frac{\Sigma, [\Gamma, \delta, [\Delta, [\Psi \vdash A] \vdash B] \vdash C] \vdash D}{\Sigma, [\Gamma, [\Delta, [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D}$$

$$\mathsf{c}{\uparrow}\ \frac{\Delta, [\Gamma, \delta \vdash A] \vdash B}{\Delta, [\Gamma, \delta, \delta \vdash A] \vdash B} \qquad \mathsf{g}{\uparrow}\ \frac{\Delta, [\Gamma \vdash A \to B] \vdash C}{\Delta, [\Gamma, A \vdash B] \vdash C} \qquad \mathsf{a}{\uparrow}\ \frac{\Delta \vdash A \to B}{\Delta, A \vdash B}$$

Figure 3: Inference rules for the system SJN

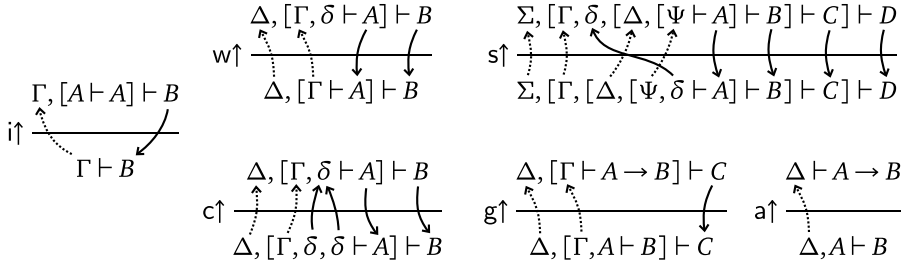
Remark 3.1. *Notice that the rule* g↑ *is really the* opposite *of* a↓*, and* a↑ *the* opposite *of* g↓*, in the sense that they just have their premise and conclusion exchanged. This is normal since the* a *and* g *rules in* JN *are performing the same action of decomposition of an implication, but in contexts of different polarities.*

The SJN system is obviously complete, since it contains the JN ∪ {e} system, for which we have proved completeness, stated in Corollary 1.19. Soundness is easy to prove, since we just have to show that the rules of the up fragment correspond to valid implications in intuitionistic logic. This is clear, since they correspond to the implications of the down fragment, reversed and used in a negative context, and if a formula $A \to B$ is provable in LJ using a proof Π_1, then for any C, the formula $(B \to C) \to A \to C$ is also provable, as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_1}{A \vdash B} \qquad \mathsf{ax}\,\dfrac{}{C \vdash C}}{\to_{\mathsf{R}}\,\dfrac{}{B \to C, A \vdash C}}}{\to_{\mathsf{L}}\,\dfrac{}{B \to C \vdash A \to C}}}{\to_{\mathsf{L}}\,\dfrac{}{\vdash (B \to C) \to A \to C}}$$

Now, we are interested in proving a property stronger than cut elimination, that we call *normalisation*: the whole up fragment is admissible for the down fragment of SJN. It is anyway required to show that we can deal with other up rules, if we want to eliminate cuts in a local way, since we introduce instances of s↑ during cut elimination. For this, we need to define an appropriate measure, taking all the up rules into account. We only consider the SJN system defined above, and no generalisation of variants of JN — they would be treated the same way.

First, we need to reconsider the definitions used in the previous section, possibly updating them to fit the generalised setting of the symmetric SJN system. The base for the tools developped to define a suitable measure for the cut elimination proof was the notion of flow-graph, which is built from the connections defined for each inference rule in the system. We can extend this definition to all up rules, to obtain the generalisation of flow-graphs for SJN.



On these extended grounds, all the definitions given in Section 2 can be trivially transfered to the symmetric setting of SJN, up to the definition of the rank of a cut instance — in particular, we define the synthetic cut rules es, ea and eg the same way, but we name them is↑, ia↑ and ig↑ to have consistent notations. At this point, we have to consider the new dynamics introduced by the up rules, which have to move upwards in the proof, and potentially create instances of other up rules along their way. Instances of these new rules thus need to have a rank, to be used in the induction performed to prove normalisation. The first step is to extend the notion of size of a rule instance, and we need a particular treatment for the dual switch.

Definition 3.2. *The* context weight *of a* s↑ *instance* r *moving some sequent* δ *is the number of atoms used in the antecedent where* δ *appears in the premise of* r.

In the scheme of the s↑ rule shown in Figure 3, this would associate a context weight calculated as the number of atoms in $\Gamma, \delta, [\Delta, [\Psi \vdash A] \vdash B]$.

Definition 3.3. *In a derivation* $\mathscr{D}$ *in* SJN*, the* size *of a cut instance* r *with a principal sequent* δ*, denoted by* $|\mathsf{r}|$*, is the number of atoms used in* δ*, the size* $|\mathsf{r}|$ *of a* s↑ *instance is the context weight of* r*, and the size* $|\mathsf{r}|$ *of an instance* r *of any other rule is* 0.

Then, we can generalise the notion of rank to all the up rules.

Definition 3.4. *In a derivation* $\mathscr{D}$ *in* SJN*, the* rank *of some instance* r *of any up rule, denoted by* $\mathscr{R}_{\mathscr{D}}(\mathsf{r})$*, is the pair* $(\mathscr{M}_{\mathscr{D}}(\mathsf{r}), |\mathsf{r}|)$*, under lexicographic ordering.*

In order to prove the local normalisation result, we will not need such complex lemmas as *weak linearisation* or *merging*. However, we need to extend the identity rule, by building the s↑ rule inside, as described in Remark 2.27, to accomodate the interaction between a standard identity i↓ rule and the s↑ rule.

Remark 3.5. *The extended form of identity* is↓*, shown below:*

$$\mathsf{is}\downarrow \frac{\Gamma, [\Delta, \Psi \vdash B] \vdash C}{\Gamma, [\Delta, [[\Psi \vdash A] \vdash A] \vdash B] \vdash C}$$

is precisely the dual of the synthetic $\mathsf{is}\uparrow$ *rule. Therefore, it contains both a down part, which is the standard* $\mathsf{i}\downarrow$ *rule, and an up part, the* $\mathsf{s}\uparrow$ *rule. This is not regarded as a problem, even if such instances remain after normalisation, because this rule has the subformula property. In fact, the equivalent of this rule was proposed for the sequent calculus [SH11], but it appears here naturally in the normalisation argument.*

We can now turn to the proof of local normalisation. As in the previous section, we will pick the topmost up rule instance to be moved upwards in the proof, and eliminate it, to show that from an arbitrary proof in SJN we can build an equivalent proof in JN, which is the *normalised* version of the original proof.

The normalisation procedure treats cut instances exactly the same way as the cut elimination procedure, except for the rewriting involved in the main cases, where a cut is reduced into two smaller cuts. In this case, the $\mathsf{s}\uparrow$ rule comes into play, and must be eliminated as well. Then, moving such an instance up is simpler than moving a cut instance — in particular, no rule instance gets blocked on this one — but it requires to use dual contractions and weakenings, to be eliminated.

Remark 3.6. *We could restrict normalisation to the elimination of cuts* $\mathsf{is}\uparrow$*,* $\mathsf{ia}\uparrow$ *and* $\mathsf{ig}\uparrow$*, dual contractions* $\mathsf{c}\uparrow$ *and weakenings* $\mathsf{w}\uparrow$*, and dual switches* $\mathsf{s}\uparrow$ *— that is, avoid dealing with the dual left and right implication rules* $\mathsf{g}\uparrow$ *and* $\mathsf{a}\uparrow$*. Indeed, the rules* $\mathsf{g}\uparrow$ *and* $\mathsf{a}\uparrow$ *are not introduced during the elimination process of other up rules.*

Theorem 3.7 (Local normalisation)**.** *Any proof* $\mathscr{D}$ *of a nested sequent* $\Gamma \vdash B$ *in* SJN *can be transformed into a proof* $\mathscr{D}'$ *of* $\Gamma \vdash B$ *in* JN*.*

Proof. If there is no up rule instance in the given proof $\mathscr{D}$, we are done, since $\mathscr{D}$ is a valid proof in the down fragment JN. In the general case, we proceed by induction on the number of up rule instances in $\mathscr{D}$, and consider the topmost such instance r in $\mathscr{D}$ with a target that we name $\mathscr{D}_2$. Then, we proceed by induction on the pair $(\mathscr{R}_{\mathscr{D}}(\mathsf{r}), \mathscr{H}(\mathscr{D}_2))$, to show that this up instance can be eliminated. For that, we use a case analysis on the bottommost instance r_1 in $\mathscr{D}_2$:

$$\mathsf{is}\uparrow / \mathsf{ia}\uparrow / \mathsf{ig}\uparrow \dfrac{\mathsf{r}_1 \dfrac{\overset{\mathscr{D}_2}{\zeta\{\upsilon\}}}{\xi\{\Gamma, \kappa \vdash B\}}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}} \qquad \textit{or} \qquad \mathsf{r}\uparrow \dfrac{\mathsf{r}_1 \dfrac{\overset{\mathscr{D}_2}{\zeta\{\upsilon\}}}{\xi\{\kappa\}}}{\xi\{\mu\}}$$

1. If r_1 is not in the scope of r, we permute it down and proceed by induction hypothesis, because the height of $\mathscr{D}_2$ has decreased.
2. If r is a cut instance, and r_1 is a switch moving material into the sequent κ introduced by the cut, or affects only the inside of κ, we can proceed as in the basic cut elimination procedure, and assimilate r_1 into the cut, so that we can use the induction hypothesis, since the height of $\mathscr{D}_2$ has decreased.

Thus, we only need to consider cases in which r_1 is blocked above the up rule instance r, and cannot be assimilated inside r if it is a cut. There are many cases to consider, and we start with the situation where r is a cut instance, which is treated the same way as in the basic cut elimination procedure.

3. If r_1 is an identity $\mathsf{is}{\downarrow}$ instance matching r, it can interact with r and disappear, and we can use the main induction hypothesis, since one cut was eliminated — the transformation we use is the same as before if the identity is applied on the positive sequent introduced by an $\mathsf{is}{\uparrow}$ cut:

$$\dfrac{\dfrac{\xi\{\Gamma, \Delta_1, A \vdash B\}}{\xi\{\Gamma, [[\Delta_1, A \vdash A] \vdash A] \vdash B\}}\,\mathsf{is}{\downarrow}}{\xi\{\Gamma, \Delta_1, A \vdash B\}}\,\mathsf{is}{\uparrow} \quad\longrightarrow\quad \xi\{\Gamma, \Delta_1, A \vdash B\}$$

but different if it is applied to the negative occurrence of such a cut:

$$\dfrac{\dfrac{\xi\{\Sigma, [\Psi, [\Delta_1 \vdash A] \vdash D] \vdash E\}}{\xi\{\Sigma, [\Psi, [[[\Delta_1 \vdash A] \vdash A] \vdash A] \vdash D] \vdash E\}}\,\mathsf{is}{\downarrow}}{\xi\{\Sigma, [\Psi, [\Delta_1 \vdash A] \vdash D] \vdash E\}}\,\mathsf{is}{\uparrow} \longrightarrow \xi\{\Sigma, [\Psi, [\Delta_1 \vdash A] \vdash D] \vdash E\}$$

In the case of an $\mathsf{ia}{\uparrow}$ instance, the body of the cut does not disappear, and as in the basic cut elimination proof we can rewrite it into a derivation $\mathscr{D}_1'$:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Sigma, \Psi \vdash E] \vdash B\}}{\xi\{\Gamma, [[[\Sigma \vdash C \to D] \vdash C \to D], \Psi \vdash E] \vdash B\}}\,\mathsf{is}{\downarrow}}{\xi\{\Gamma, [\Sigma \vdash C \to D], \Delta_2 \vdash B\}}\,\mathsf{ia}{\uparrow} \quad\longrightarrow\quad \dfrac{\begin{array}{c}\xi\{\Gamma, [\Sigma, \Psi \vdash E] \vdash B\} \\ \mathscr{D}_1' \| \\ \xi\{\Gamma, [\Sigma, C \vdash D], \Delta_2 \vdash B\}\end{array}}{\xi\{\Gamma, [\Sigma \vdash C \to D], \Delta_2 \vdash B\}}\,\mathsf{a}{\downarrow}$$

and then we can apply the main induction hypothesis on the resulting proof, since one cut instance was erased. In the case of an $\mathsf{ig}{\uparrow}$ instance, we proceed in a similar way:

$$\dfrac{\dfrac{\xi\{\Psi\}}{\xi\{[\Psi \vdash C \to D] \vdash C \to D\}}\,\mathsf{is}{\downarrow}}{\xi\{\Delta_1, \Delta_2 \vdash C \to D\}}\,\mathsf{ig}{\uparrow} \quad\longrightarrow\quad \dfrac{\begin{array}{c}\xi\{\Psi\} \\ \mathscr{D}_1' \| \\ \xi\{\Delta_1, \Delta_2, C \vdash D\}\end{array}}{\xi\{\Delta_1, \Delta_2 \vdash C \to D\}}\,\mathsf{g}{\downarrow}$$

4. If r_1 is a $\mathsf{w}{\downarrow}$ instance erasing the principal sequent of the cut r or a $\mathsf{c}{\downarrow}$ instance duplicating it, we have to introduce weakenings or contractions respectively, and erase or duplicate the cut, as shown below in one case of weakening:

$$\dfrac{\dfrac{\xi\{\Gamma \vdash B\}}{\xi\{\Gamma, \mu \vdash B\}}\,\mathsf{w}{\downarrow}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{is}{\uparrow} / \mathsf{ia}{\uparrow} / \mathsf{ig}{\uparrow} \quad\longrightarrow\quad \dfrac{\xi\{\Gamma \vdash B\}}{\xi\{\Gamma, \Delta \vdash B\}}\,\mathsf{w}{\downarrow}^*$$

and similarly in one case of contraction:

$$\dfrac{\dfrac{\xi\{\Gamma, \kappa, \kappa \vdash B\}}{\xi\{\Gamma, \kappa \vdash B\}}\,\mathsf{c}{\downarrow}}{\xi\{\Gamma, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{is}{\uparrow} / \mathsf{ia}{\uparrow} / \mathsf{ig}{\uparrow} \quad\longrightarrow\quad \dfrac{\dfrac{\xi\{\Gamma, \kappa, \kappa \vdash B\}}{\xi\{\Gamma, \kappa, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{is}{\uparrow} / \mathsf{ia}{\uparrow} / \mathsf{ig}{\uparrow}}{\xi\{\Gamma, \Delta_1, \Delta_2, \Delta_1, \Delta_2 \vdash B\}}\,\mathsf{c}{\downarrow}^*$$

Notice that we can apply the main induction hypothesis because one cut was erased, in the first case. In the second case, we can use the other induction hypothesis on the topmost copy of the cut, because its multiplicity is smaller than the multiplicity of the original cut, and do the same with the bottommost copy, for the same reason as in the basic cut elimination procedure — finally, we use the main induction hypothesis since one cut was erased.

5. If r_1 is an $\mathsf{a}{\downarrow}$ instance and r is an instance of $\mathsf{ig}{\uparrow}$, or if r_1 is a $\mathsf{g}{\downarrow}$ instance and r is an instance $\mathsf{ia}{\uparrow}$, we have one of the situations described below:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma,[[\Delta_1,\Psi\vdash E],C\vdash D]\vdash B\}}{\xi\{\Gamma,[[\Delta_1,\Psi\vdash E]\vdash C\to D]\vdash B\}}\,\mathsf{a}{\downarrow}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}\,\mathsf{ig}{\uparrow} \qquad \dfrac{\dfrac{\dfrac{\xi\{\Gamma,[[\Delta_1,C\vdash D],\Psi\vdash E]\vdash B\}}{\xi\{\Gamma,[[\Delta_1\vdash C\to D],\Psi\vdash E]\vdash B\}}\,\mathsf{g}{\downarrow}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}\,\mathsf{ia}{\uparrow}$$

and we can permute r_1 down to the bottom of the body of the cut, so that we obtain the proofs shown below, on the left in the first case and on the right in the second case, where $\mathscr{D}_1'$ is in both case obtained by the permutation of the instance r_1 through $\mathscr{D}_1$, and $\mathscr{D}_2'$ is the topmost part of $\mathscr{D}_2$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\xi\{\Gamma,[[\Delta_1,\Psi\vdash E],C\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta_1,\Delta_2,[[C\vdash D],C\vdash D]\vdash B\}}\,\mathsf{g}{\downarrow}}{\xi\{\Gamma,\Delta_1,\Delta_2,[C\to D,C\vdash D]\vdash B\}}\,\mathsf{a}{\downarrow}}{\xi\{\Gamma,\Delta_1,\Delta_2,[C\to D\vdash C\to D]\vdash B\}}\,\mathsf{i}{\uparrow}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\xi\{\Gamma,[[\Delta_1,C\vdash D],\Psi\vdash E]\vdash B\}}{\xi\{\Gamma,\Delta_1,\Delta_2,[[C\vdash D],C\vdash D]\vdash B\}}\,\mathsf{a}{\downarrow}}{\xi\{\Gamma,\Delta_1,\Delta_2,[[C\vdash D]\vdash C\to D]\vdash B\}}\,\mathsf{g}{\downarrow}}{\xi\{\Gamma,\Delta_1,\Delta_2,[C\to D\vdash C\to D]\vdash B\}}\,\mathsf{i}{\uparrow}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}$$

Then, we can rewrite each of these proofs into another proof where the cut has been splitted into to cuts of smaller size, as in the basic cut elimination procedure. However, because of the available up rules, there is no need for a *merging* lemma, and we can glue the parts together using an instance of $\mathsf{s}{\uparrow}$, so that above the new cuts we can use $\mathscr{D}_3$, the composition of $\mathscr{D}_1'$ and $\mathscr{D}_2'$:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma,\Delta_1,\Delta_2,[[C\vdash D],C\vdash D]\vdash B\}}{\xi\{\Gamma,\Delta_1,\Delta_2,[[[C\vdash C]\vdash D]\vdash D]\vdash B\}}\,\mathsf{s}{\uparrow}}{\xi\{\Gamma,\Delta_1,\Delta_2,[D\vdash D]\vdash B\}}\,\mathsf{i}{\uparrow}}{\xi\{\Gamma,\Delta_1,\Delta_2\vdash B\}}\,\mathsf{i}{\uparrow}$$

We can then use the induction hypothesis on the dual switch $\mathsf{s}{\uparrow}$ instance, because its size is necessarily smaller than the size of the original cut. Finally, we use the induction hypothesis on the two cuts below as well, since they are of smaller size than the original, and we use the main induction hypothesis in the end, because one cut was eliminated.

The remaining cases are the new cases introduced by the use of the symmetric system, with a dual up fragment. Because of the previous case of reduction of one cut into two smaller ones, we have to show how to eliminate all s↑ instances in a given proof — what the elimination of this *glueing* s↑ instance basically does is the implementation of the merging lemma *within* the syntactic dynamics of the logic.

Given any up rule instance r, the transformation to perform, when the instance r_1 above is a weakening or a contraction erasing or duplicating the whole sequent that was change by r, is always the same. In such a case, we treat r the same way as we treated the situation of a cut used below a structural rule:

6. If r_1 is a weakening w↓ instance erasing the whole sequent affected by r, we just have to erase r, and we can go on using the main induction hypothesis, since one up rule instance was eliminated:

$$\mathsf{w}\downarrow\dfrac{\mathsf{r}\dfrac{\xi\{\Gamma\vdash B\}}{\xi\{\Gamma,\kappa\vdash B\}}}{\xi\{\Gamma,\mu\vdash B\}}\quad\longrightarrow\quad\mathsf{w}\downarrow\dfrac{\xi\{\Gamma\vdash B\}}{\xi\{\Gamma,\mu\vdash B\}}$$

7. If r_1 is a contraction c↓ instance duplicating the sequent affected by r, we can duplicate r and go on by induction hypothesis, which is possible since after this transformation, r has a smaller multiplicity than the original:

$$\mathsf{c}\downarrow\dfrac{\mathsf{r}\dfrac{\xi\{\Gamma,\kappa,\kappa\vdash B\}}{\xi\{\Gamma,\kappa\vdash B\}}}{\xi\{\Gamma,\mu\vdash B\}}\quad\longrightarrow\quad\mathsf{r}\dfrac{\mathsf{r}\dfrac{\xi\{\Gamma,\kappa,\kappa\vdash B\}}{\xi\{\Gamma,\kappa,\mu\vdash B\}}}{\mathsf{c}\downarrow\dfrac{\xi\{\Gamma,\mu,\mu,\vdash B\}}{\xi\{\Gamma,\mu\vdash B\}}}$$

In other cases, either the permutation is a trivial one, or we provide a rewriting to apply in this situation. We start with the rewritings devoted to the elimination of all the dual switch s↑ instances, which requires the introduction of the other up rules w↑ and c↑:

8. If r_1 is an identity is↓ instance matching the s↑ instance r, we integrate the dual switch inside the identity, and go on using the main induction hypothesis since we have removed one up rule instance:

$$\mathsf{i}\downarrow\dfrac{\mathsf{s}\uparrow\dfrac{\xi\{\Gamma,[\Delta,\delta,\Psi\vdash B]\vdash C\}}{\xi\{\Gamma,[\Delta,\delta,[[\Psi\vdash A]\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,[\Delta,[[\delta,\Psi\vdash A]\vdash A]\vdash B]\vdash C\}}\quad\longrightarrow\quad\mathsf{is}\downarrow\dfrac{\xi\{\Gamma,[\Delta,\delta,\Psi\vdash B]\vdash C\}}{\xi\{\Gamma,[\Delta,[[\delta,\Psi\vdash A]\vdash A]\vdash B]\vdash C\}}$$

9. If r_1 is a weakening $\mathsf{w}\downarrow$ instance matching the dual switch, we have to turn the $\mathsf{s}\uparrow$ instance into a dual weakening $\mathsf{w}\uparrow$ instance, shown below, and we can go on by induction hypothesis, since the new $\mathsf{w}\uparrow$ instance has exactly the same multiplicity as the original r instance, but the height of $\mathscr{D}_2$ has decreased:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Delta, \delta, [\Sigma \vdash B] \vdash C] \vdash D\}}{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{w}\downarrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{s}\uparrow \longrightarrow \dfrac{\dfrac{\xi\{\Gamma, [\Delta, \delta, [\Sigma \vdash B] \vdash C] \vdash D\}}{\xi\{\Gamma, [\Delta, [\Sigma \vdash B] \vdash C] \vdash D\}}\,\mathsf{w}\uparrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{w}\downarrow$$

10. If r_1 is a contraction $\mathsf{c}\downarrow$ instance matching the dual switch, the $\mathsf{s}\uparrow$ instance is duplicated when it permutes with the contraction, so that from:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi \vdash A], [\Psi \vdash A] \vdash B] \vdash C] \vdash D\}}{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{c}\downarrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{s}\uparrow$$

we obtain the new situation shown below, where a dual contraction instance $\mathsf{c}\uparrow$ has been created to glue the proof above r_1 to the premise of the two dual switches, by collapsing the two copies of δ created:

$$\dfrac{\dfrac{\dfrac{\dfrac{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi \vdash A], [\Psi \vdash A] \vdash B] \vdash C] \vdash D\}}{\xi\{\Gamma, [\Delta, \delta, \delta, [\Sigma, [\Psi \vdash A], [\Psi \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{c}\uparrow}{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi \vdash A], [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{s}\uparrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, \delta \vdash A], [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{s}\uparrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, \delta \vdash A] \vdash B] \vdash C] \vdash D\}}\,\mathsf{c}\downarrow$$

We can use the induction hypothesis on the dual contraction $\mathsf{c}\uparrow$ introduced, since its multiplicity is at most the same as the multiplicity of r — because δ is in positive position. Then, we can also use the induction hypothesis on the two copies of the $\mathsf{s}\uparrow$ instance, since they have a smaller multiplicity than r, and we use the main induction hypothesis, as one up rule was eliminated.

11. If r_1 is a switch $\mathsf{s}\downarrow$ instance, there are two possible configurations in which the $\mathsf{s}\uparrow$ instance can interact, the first one happening when r_1 moves material inside the material moved out by the $\mathsf{s}\uparrow$ instance, so that:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Delta, [\Phi, \delta \vdash A], [\Sigma, [\Psi \vdash B] \vdash C] \vdash D] \vdash E\}}{\xi\{\Gamma, \delta, [\Delta, [\Phi \vdash A], [\Sigma, [\Psi \vdash B] \vdash C] \vdash D] \vdash E\}}\,\mathsf{s}\downarrow}{\xi\{\Gamma, \delta, [\Delta, [\Sigma, [\Psi, [\Phi \vdash A] \vdash B] \vdash C] \vdash D] \vdash E\}}\,\mathsf{s}\uparrow$$

is rewritten into the derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma, [\Delta, [\Phi, \delta \vdash A], [\Sigma, [\Psi \vdash B] \vdash C] \vdash D] \vdash E\}}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, [\Phi, \delta \vdash A] \vdash B] \vdash C] \vdash D] \vdash E\}}\,\mathsf{s}\uparrow}{\xi\{\Gamma, [\Delta, [\Sigma, \delta, [\Psi, [\Phi \vdash A] \vdash B] \vdash C] \vdash D] \vdash E\}}\,\mathsf{s}\downarrow}{\xi\{\Gamma, \delta, [\Delta, [\Sigma, [\Psi, [\Phi \vdash A] \vdash B] \vdash C] \vdash D] \vdash E\}}\,\mathsf{s}\downarrow$$

and we can conclude by induction hypothesis, since the size of the s↑ instance has decreased. The other configuration is symmetric to the first one, where the s↑ instance moves material out of a sequent before r_1 moves this sequent inside another, so that from:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi, [\Xi, [\Phi \vdash A] \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Phi \vdash A], [\Psi, [\Xi \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}\,\mathsf{s}\downarrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Phi, \delta \vdash A], [\Psi, [\Xi \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}\,\mathsf{s}\uparrow$$

we obtain the new derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma, [\Delta, \delta, [\Sigma, [\Psi, [\Xi, [\Phi \vdash A] \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, \delta, [\Xi, [\Phi \vdash A] \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}\,\mathsf{s}\uparrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Psi, [\Xi, [\Phi, \delta \vdash A] \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}\,\mathsf{s}\uparrow}{\xi\{\Gamma, [\Delta, [\Sigma, [\Phi, \delta \vdash A], [\Psi, [\Xi \vdash B] \vdash C] \vdash D] \vdash E] \vdash F\}}\,\mathsf{s}\downarrow$$

and we can use the induction hypothesis on the topmost s↑ instance since it has the same size of the original and the height of $\mathscr{D}_2$ has decreased. We can then use the induction hypothesis again, on the other switch, because its size is smaller than that of the original.

All other cases involving s↑ instances are trivial permutations. Then, we show how to treat the instances of the dual weakening w↑ rule, and this will never require to introduce instances of other rules than w↑ itself — similarly to the treatment of weakening in the down fragment, moving the dual weakening rule is mostly about erasing the rule instances that are encountered.

12. If r_1 only affects a sequent contained inside the sequent introduced by r, it can simply be erased — this is dual to the situation where an up rule instance is erased by a weakening — and we can go on by induction hypothesis since the height of $\mathscr{D}_2$ has decreased.

13. If r_1 is an is↓ instance applied exactly on the sequent introduced by r, both rule instances can be erased, and we can go on by induction hypothesis, since one up rule instance was eliminated:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Delta \vdash B] \vdash C\}}{\xi\{\Gamma, [\Delta, [[\Psi \vdash A] \vdash A] \vdash B] \vdash C\}}\,\mathsf{is}\downarrow}{\xi\{\Gamma, [\Delta \vdash B] \vdash C\}}\,\mathsf{w}\uparrow \quad\longrightarrow\quad \xi\{\Gamma, [\Delta \vdash B] \vdash C\}$$

14. If r_1 is an instance of s↓ moving material inside the sequent introduced by r, we can erased this material and introduce it directly using a dual weakening, and then use the induction hypothesis, which is of course possible since the height of $\mathscr{D}_2$ has decreased:

$$\dfrac{\dfrac{\xi\{\Gamma, [\Delta, [\Psi, \delta \vdash A] \vdash B] \vdash C\}}{\xi\{\Gamma, \delta, [\Delta, [\Psi \vdash A] \vdash B] \vdash C\}}\,\mathsf{s}\downarrow}{\xi\{\Gamma, \delta, [\Delta \vdash B] \vdash C\}}\,\mathsf{w}\uparrow \quad\longrightarrow\quad \dfrac{\dfrac{\xi\{\Gamma, [\Delta, [\Psi, \delta \vdash A] \vdash B] \vdash C\}}{\xi\{\Gamma, [\Delta, \vdash B] \vdash C\}}\,\mathsf{w}\uparrow}{\xi\{\Gamma, \delta, [\Delta \vdash B] \vdash C\}}\,\mathsf{w}\downarrow$$

All other cases involving $\mathsf{w}\uparrow$ instances are trivial permutations. Then, we show how to eliminate instances of the dual contraction $\mathsf{c}\uparrow$ rule, and this is similar to the situation described for dual weakenings — when the dual contraction is moved upwards, it forces the duplication of other rule instances.

15. If r_1 only affects a sequent inside the sequent resulting from the collapse of the two δ sequents operated by the r, it can be duplicated — this is dual to the situation where an up instance is duplicated by a contraction — and we can use the induction hypothesis since the height of $\mathscr{D}_2$ has decreased.

16. If r_1 is an instance of $\mathsf{s}\downarrow$ moving material inside the sequent resulting from the collapse operated by r, we can use a contraction to duplicate the sequent δ and then use two copies of r_1 before we collapse the resulting sequents with the dual contraction — and we can use the induction hypothesis, because the height of $\mathscr{D}_2$ has decreased — so that:

$$\mathsf{c}\uparrow \dfrac{\mathsf{s}\downarrow \dfrac{\xi\{\Gamma,[\Delta,[\Psi,\delta\vdash A]\vdash B]\vdash C\}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A],[\Psi\vdash A]\vdash B]\vdash C\}}$$

is rewritten into the derivation:

$$\dfrac{\mathsf{s}\downarrow \dfrac{\mathsf{s}\downarrow \dfrac{\mathsf{c}\uparrow \dfrac{\xi\{\Gamma,[\Delta,[\Psi,\delta\vdash A]\vdash B]\vdash C\}}{\xi\{\Gamma,[\Delta,[\Psi,\delta\vdash A],[\Psi,\delta\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A],[\Psi,\delta\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,\delta,[\Delta,[\Psi\vdash A],[\Psi\vdash A]\vdash B]\vdash C\}}}{\xi\{\Gamma,\delta,[\Delta,[\Psi\vdash A],[\Psi\vdash A]\vdash B]\vdash C\}}\ \mathsf{c}\downarrow$$

All other cases involving $\mathsf{c}\uparrow$ instances are trivial permutations. Then, we show how to eliminate instances of the $\mathsf{g}\uparrow$ rule, which is quite easy and never requires to introduce other up rule instances:

17. If r_1 is an $\mathsf{is}\downarrow$ instance involving the implication formed by r, we can apply the $\mathsf{g}\downarrow$ rule on the other implication and use the following transformation to remove r — and we go on using the main induction hypothesis, since one up rule instance was eliminated — so that the derivation:

$$\mathsf{g}\uparrow \dfrac{\mathsf{is}\downarrow \dfrac{\xi\{\Gamma,[\Delta,\Psi\vdash C]\vdash D\}}{\xi\{\Gamma,[\Delta,[[\Psi\vdash A\to B]\vdash A\to B]\vdash C]\vdash D\}}}{\xi\{\Gamma,[\Delta,[[\Psi,A\vdash B]\vdash A\to B]\vdash C]\vdash D\}}$$

is rewritten into:

$$\mathsf{g}\downarrow \dfrac{\mathsf{s}\downarrow \dfrac{\mathsf{i}\downarrow \dfrac{\mathsf{is}\downarrow \dfrac{\xi\{\Gamma,[\Delta,\Psi\vdash B]\vdash C\}}{\xi\{\Gamma,[\Delta,[[\Psi\vdash B]\vdash B]\vdash C]\vdash D\}}}{\xi\{\Gamma,[\Delta,[[\Psi,[A\vdash A]\vdash B]\vdash B]\vdash C]\vdash D\}}}{\xi\{\Gamma,[\Delta,[[\Psi,A\vdash B],A\vdash B]\vdash C]\vdash D\}}}{\xi\{\Gamma,[\Delta,[[\Psi,A\vdash B]\vdash A\to B]\vdash C]\vdash D\}}$$

18. If r_1 is an $\mathsf{a}{\downarrow}$ instance decomposing the implication formed by r, then both instances can be removed, and we use the main induction hypothesis, since one up rule instance was eliminated:

$$\dfrac{\dfrac{\xi\{\Gamma,[\Delta,A\vdash B]\vdash C\}}{\xi\{\Gamma,[\Delta\vdash A\to B]\vdash C\}}\,\mathsf{a}{\downarrow}}{\xi\{\Gamma,[\Delta,A\vdash B]\vdash C\}}\,\mathsf{g}{\uparrow} \quad\longrightarrow\quad \xi\{\Gamma,[\Delta,A\vdash B]\vdash C\}$$

All other cases involving $\mathsf{g}{\uparrow}$ instances are trivial permutations. Then, we show how to eliminate instances of the dual application $\mathsf{a}{\uparrow}$ rule, which is almost the same as for the $\mathsf{g}{\uparrow}$ rule, since these rules are performing the same action, in contexts of different polarities.

19. If r_1 is an $\mathsf{is}{\downarrow}$ instance involving the implication formed by r, we can apply the $\mathsf{a}{\downarrow}$ rule on the other implication and use the following transformation to remove r — and we go on using the main induction hypothesis, since one up rule instance was eliminated — so that:

$$\dfrac{\dfrac{\xi\{\Gamma,[\Delta,\Psi\vdash C]\vdash D\}}{\xi\{\Gamma,[\Delta,[[\Psi\vdash A\to B]\vdash A\to B]\vdash C]\vdash D\}}\,\mathsf{is}{\downarrow}}{\xi\{\Gamma,[\Delta,[[\Psi\vdash A\to B],A\vdash B]\vdash C]\vdash D\}}\,\mathsf{a}{\uparrow}$$

is rewritten into the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\xi\{\Gamma,[\Delta,\Psi\vdash B]\vdash C\}}{\xi\{\Gamma,[\Delta,[[\Psi\vdash B]\vdash B]\vdash C]\vdash D\}}\,\mathsf{is}{\downarrow}}{\xi\{\Gamma,[\Delta,[[\Psi,[A\vdash A]\vdash B]\vdash B]\vdash C]\vdash D\}}\,\mathsf{i}{\downarrow}}{\xi\{\Gamma,[\Delta,[[\Psi,A\vdash B],A\vdash B]\vdash C]\vdash D\}}\,\mathsf{s}{\downarrow}}{\xi\{\Gamma,[\Delta,[[\Psi\vdash A\to B],A\vdash B]\vdash C]\vdash D\}}\,\mathsf{a}{\downarrow}$$

20. If r_1 is an $\mathsf{g}{\downarrow}$ instance decomposing the implication formed by r, then both instances can be removed, and we use the main induction hypothesis, since one up rule instance was eliminated:

$$\dfrac{\dfrac{\xi\{\Gamma,A\vdash B\}}{\xi\{\Gamma\vdash A\to B\}}\,\mathsf{g}{\downarrow}}{\xi\{\Gamma,A\vdash B\}}\,\mathsf{a}{\uparrow} \quad\longrightarrow\quad \xi\{\Gamma,A\vdash B\}$$

All other cases involving $\mathsf{a}{\uparrow}$ instances are trivial permutations. This case analysis describes the rewritings to be applied to remove all up rule instances, and when the inductions come to an end, the resulting proof does not contain any such instance anymore, so that it is a valid proof in JN. □

Chapter 4

Intuitionistic Logic in the Calculus of Structures

This chapter elaborates on the systems previously introduced for intuitionistic logic in nested sequents, by describing two presentations of intuitionistic logic in the plain formalism of the calculus of structures, where all deduction is collapsed into one level and the proof construction process is reduced to the rewriting of formulas. In both cases, the emphasis is put on the impact of the system design on the proof normalisation transformation.

The first presentation follows the style of the sequent calculus, but implements the collapse of the logical level and the meta-level by retaining only the implication connective inside structures that can be interpreted either as sequents or formulas. In such a system, there are less inference rules, since for example, the *curryfication* operation relating $\Gamma \vdash A \to B$ to $\Gamma, A \vdash B$ is implicit. In the cut elimination process, this simplifies the situation by eliminating the need to have instances of both left and right implication rules above a cut to decompose it. In particular, it allows to obtain a common property of many systems in the calculus of structures: the ability to reduce the cut to its atomic form by a direct decomposition rather than through the permutative procedure for cut elimination. However, this leaves the problem found in the procedure defined for nested sequents, as its equivalent in the calculus of structures also relies on a non-local rewriting of proofs. The generalisation into a symmetric system is also presented, following the same principles as in nested sequents, and the local normalisation procedure is adapted to this setting.

The second presentation follows the style of natural deduction, where a system is designed by pairs of introduction and elimination rules. This is quite unusual for the setting of the calculus of structures, but it allows for a simpler treatment of the normalisation of proofs. Indeed, the elimination of *detours*, formed by sequences of introduction and elimination instances on the same connective, is performed in such a system the same way as in natural deduction, using a cut and a permutative procedure — the substitutive procedure would be difficult to implement here, since branches are interleaved into the flat structure of proofs. In particular, we present a completely local rewriting procedure to transform an intuitionistic proof into a normal one, directly inspired from the standard permutative procedure.

1 A System in Sequent Style

We present a proof system for the purely implicative fragment of intuitionistic logic in the calculus of structures, which uses the standard way of representing proofs in the setting of *deep inference* [Gug07] by defining only one level for reasoning: the level of formulas — by opposition to the traditional representation of proofs in the sequent calculus, using in addition a *meta-level* where the sequents are informally equivalent to logical formula. However, this system is in *sequent style* in the sense that it comes with a cut rule that can be eliminated to produce from any proof a normal proof where all formulas are built with atoms that are, informally speaking, already present in its conclusion — this corresponds to the *subformula property*.

This system is almost identical to the system $\mathsf{JN} \cup \{\mathsf{e}\}$ presented in the previous chapter, but it removes the need for the rules relating the logical implication $\rightarrow$ and the meta-level implication represented by the $\vdash$ symbol in sequents, and handles logical equivalences through a congruence relation on formulas, as usually done in the calculus of structures. Therefore, establishing its properties will be easy, since the techniques developped for $\mathsf{JN} \cup \{\mathsf{e}\}$ can be adapted in this new setting.

1.1 Basic Definitions

As in the systems we defined in nested sequents, we assume given a countable set of *atoms*, denoted by small latin letters such as a, b, c, and the connective $\rightarrow$ for intuitionistic implication. We will here use a unit, denoted by $\top$, which represents truth and will therefore be the left unit of the implication[1].

Definition 1.1. *The* formulas *of intuitionistic logic are defined by the grammar:*

$$A, B \ ::= \ a \mid \top \mid A \rightarrow B$$

Then, as usual in the calculus of structures, we consider logical formulas through a set of equations forming a congruence, which defines equivalence classes that are the objects we will actually deal with. The purpose of this congruence is to simplify notations: because some rewritings are incorporated in the congruence, we need less inference rules, and when writing proofs, the reading is made easier by the absence of such tedious steps, which usually do not convey the idea of the proof.

Definition 1.2. *The* structures *of our system are defined as the equivalence classes of formulas generated by the congruence described by the equations shown in Figure 1.*

The first of these equations makes $\top$ the left unit of the implication, and the second one allows to exchange two formulas on the left, which would correspond to the ability of exchanging formulas in the antecedent of a sequent — an antecedent being a multiset. In the following, we sometimes make explicit the rewriting steps corresponding to the congruence, by using a general inference rule $\equiv$ which can be applied in an instance with premise A and conclusion B whenever $A \equiv B$. Notice that the implication connective is right associative, so that we can write a structure $A \rightarrow (B \rightarrow C)$ in the simpler form $A \rightarrow B \rightarrow C$ without ambiguity.

[1] This unit is useful to deal with the equivalent of sequent of the shape $\vdash A$, with an empty antecedent.

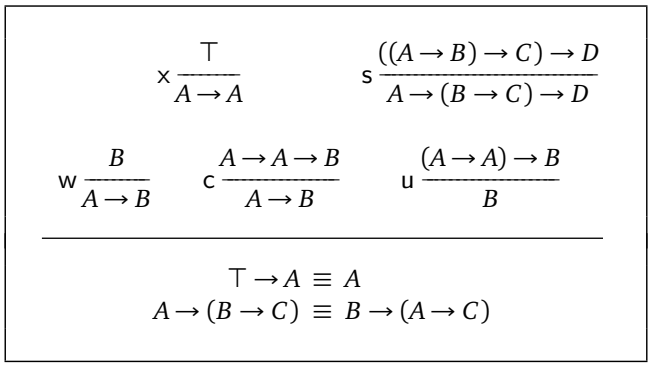


Figure 1: Inference rules and congruence for the system $\mathsf{JS} \cup \{\mathsf{u}\}$

As usual in a deep inference setting, we have the ability to apply inference rules inside a context, and here this means rewriting a structure deep inside another structure. As in the nested sequents systems, we simplify notations by considering only the *positive* contexts, which are those located on the left-hand side of an even number of implications.

Definition 1.3. *The* contexts *of our system are structures with a hole* $\{\ \}$ *meant to be filled by another structure, and are defined by the following grammar:*

$$\xi \ ::= \ \{\ \} \mid (\xi \to A) \to B$$

Remark 1.4. *As in the nested sequent systems, everything is considered here from the viewpoint of positive contexts, which preserve polarity, so that a context plugged in a negative position has a hole in negative position.*

Contexts will be denoted as $\xi\{\ \}$ or $\zeta\{\ \}$, so that for example $\xi\{A\}$ is the context ξ where the hole has been replaced by the structure A. The definition of contexts can be extended to several holes easily, such a context being denoted by $\xi\{\ \}^+$ and allowing notations such as the following:

$$\xi_i\{A_i\}^+ \ = \ \xi\{A_1\}\cdots\{A_n\} \qquad \text{for some } n \in \mathbb{N} \tag{9}$$

Inference rule instances are, also here, defined through two structures schemes that can be instantiated and can always be plugged inside a context.

Notice that in the approach of the calculus of structures, where the meta-level has been completely absorbed into the definition of structures, and thus collapsed with with logical level, inferences rules can be applied anywhere inside structures at any point in a derivation, even in the conclusive structure. This is different from the situation of nested sequents, where a subformula is made accessible only after the formula around it has been decomposed into the meta-level.

The specific set of inference rules used in our sequent style system, that we call here $\mathsf{JS} \cup \{\mathsf{u}\}$, is given in Figure 1. The *cut-free* system JS is obtained by removing the rule u from this set — this is the rule corresponding to the usual cut rule.

The presentation of the system JS differs, at first sight, from the usual sequent calculus LJ, because of the collapse of the logical level and the meta-level, and due to the decomposition allowed in a deep inference setting. But the weakening and contraction rules w and c are standard, as well as the axiom rule x, and the cut rule u corresponds to the traditional cut rule when considering the inner occurrence of A as the conclusion of the branch proving the cut formula.

As in the case of nested sequents, one of the key features of this deep inference system is the *switch* rule s, which allows to perform the equivalent of the context splitting operation from the sequent calculus, in a lazy way. For example, in the cut rule u, no structures are given as hypothesis to prove the cut structure A, because they can be moved inside later in the proof construction, using a switch. Notice that there is no equivalent of the left/right implication rules $\rightarrow_{\mathsf{L}}$ and $\rightarrow_{\mathsf{R}}$ here, because they are only required to deal with the decomposition of a logical implication into a meta-level implication, but the switch rule is also required to perform the task of the $\rightarrow_{\mathsf{L}}$ rule.

Example 1.5. *Consider the two following simple derivations, which illustrate the use of the axiom and cut rules, as well as the switch and the equations on formulas:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\top \rightarrow A) \rightarrow A}{(((A \rightarrow A) \rightarrow A) \rightarrow A) \rightarrow A)\rightarrow A}\,\mathsf{x}}{((A \rightarrow (A \rightarrow A) \rightarrow A) \rightarrow A) \rightarrow A}\,\mathsf{s}}{A \rightarrow (((A \rightarrow A) \rightarrow A) \rightarrow A) \rightarrow A}\,\mathsf{s}}{A \rightarrow (A \rightarrow A) \rightarrow A}\,\mathsf{u}}{A \rightarrow A}\,\mathsf{u} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\top}{A \rightarrow A}\,\mathsf{x}}{(\top \rightarrow A) \rightarrow A}\,\equiv}{((A \rightarrow A) \rightarrow A) \rightarrow A}\,\mathsf{x}}{A \rightarrow (A \rightarrow A) \rightarrow A}\,\mathsf{s}}{A \rightarrow A}\,\mathsf{u}$$

where we see how the switch is responsible for moving material to the location where it is needed. On the right, one switch is needed to place the A where it can be used by the axiom rule, but on the left A must be pushed twice. Notice that the shape of the x *rule imposes a certain order on the application of its instances: we must apply it inside first. In some systems [BM08], a variant axiome rule can be used:*

$$\dfrac{B}{(B \rightarrow A) \rightarrow A}$$

where the structure B to be proved to validate the assumption A is carried over to the premise, as it needs to be proved, for validating the whole deduction. The equivalent of this rule has also been suggested in the sequent calculus [SH11].

It appears now that the differences between the $\mathsf{JS} \cup \{\mathsf{u}\}$ system and the nested sequents system $\mathsf{JN} \cup \{\mathsf{e}\}$ are minimal, and mostly related to the notations rather than to the intrisic representation of intuitionistic proofs. For instance, a proof of $\mathsf{JS} \cup \{\mathsf{u}\}$ starting by rewriting a substructure deep inside the complete conclusion can be simulated in $\mathsf{JN} \cup \{\mathsf{e}\}$ by first decomposing the formula into the meta-level, until the target subformula is made accessible. These additional steps correspond to invisible steps in $\mathsf{JS} \cup \{\mathsf{u}\}$, because of the logical equivalence of the connectives and the meta-level structures. In systems with built-in structural rules, the situation is slightly more complicated.

Example 1.6. *Here are the translations of the proofs given as examples for the nested sequents system* $\mathsf{JN} \cup \{\mathsf{e}\}$ *in the system* $\mathsf{JS} \cup \{\mathsf{u}\}$*. Written as below, without the explicit congruence steps, they are a little simpler than their nested sequents counterparts.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\top}{B \to B}\,\mathsf{x}}{((A \to A) \to B) \to B}\,\mathsf{x}}{(A \to B) \to A \to B}\,\mathsf{s}}{((A \to A) \to A \to B) \to A \to B}\,\mathsf{x}}{(A \to A \to B) \to A \to A \to B}\,\mathsf{s}}{(A \to A \to B) \to A \to B}\,\mathsf{c}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\top}{B \to B}\,\mathsf{x}}{((B \to B) \to B) \to B}\,\mathsf{x}}{((((B \to B) \to B) \to B) \to B) \to B}\,\mathsf{s}}{((B \to (B \to B) \to B) \to B) \to B}\,\mathsf{s}}{B \to (((B \to B) \to B) \to B) \to B}\,\mathsf{w}}{B \to (((B \to B) \to A \to B) \to B) \to B}\,\mathsf{u}}{B \to ((A \to B) \to B) \to B}$$

We can of course show that it is always possible to reduce the axiom rule to the atomic form, by replacing a general instance with a derivation using atomic axioms and switch instances to dispatch the different atoms to the right places.

Proposition 1.7. *Any instance of the* x *rule can be replaced by a derivation in* JS *with same premise and conclusion, using instances of* x *only in the atomic form.*

Proof. By induction on the structure A affected by a general instance of the axiom rule, with premise $\xi\{\top\}$ and conclusion $\xi\{A \to A\}$. If A is an atom a, then this axiom is already in atomic form and we are done. In the general case, A is an implication $B \to C$, and we replace the initial instance by the following derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\top\}}{\xi\{C \to C\}}\,\mathsf{x}}{\xi\{((B \to B) \to C) \to C\}}\,\mathsf{x}}{\xi\{(B \to C) \to B \to C\}}\,\mathsf{s}$$

to which we can apply the induction hypothesis. □

1.2 Correspondence to the Sequent Calculus

The simplest way to prove that our system is suitable for intuitionistic logic is to prove soundness and completeness with respect to the sequent calculus. We use the variant $\mathsf{LJ}_{\top} \cup \{\mathsf{cut}\}$ shown below in Figure 2, similar to the one we have used in the previous chapter, but incorporating the truth unit. This requires translations between the structures and sequents, which relies on the correspondence between sequents and formulas. Translating some structure into a sequent is easy, since we simply have to pick one formula in the equivalence class that defines the structure, and use it in a sequent. The other way around, we translate the same way nested sequents were translated into formulas.

Remark 1.8. *To make the translation from structures to sequents deterministic, we assume given a total order on atoms, allowing us to choose a formula accordingly.*

$$\mathsf{ax}\,\frac{}{A \vdash A} \qquad \mathsf{cut}\,\frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B} \qquad \mathsf{weak}\,\frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

$$\rightarrow_{\mathsf{R}}\,\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \qquad \rightarrow_{\mathsf{L}}\,\frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \rightarrow B \vdash C} \qquad \mathsf{cont}\,\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}$$

$$\top_{\mathsf{R}}\,\frac{}{\vdash \top} \qquad \top_{\mathsf{L}}\,\frac{\Gamma \vdash A}{\Gamma, \top \vdash A}$$

Figure 2: Inference rules for system $\mathsf{LJ}_{\top} \cup \{\mathsf{cut}\}$

The correspondence between formulas of the intuitionistic sequent calculus and formulas and structures defined in our setting is immediate, we can define it as the identity transformation.

The translation from sequents to structures is performed by a simple induction on the shape of the sequent, so that any given sequent is turned into the logical formula it is equivalent to, and this formula is expressed as a structure to be used in the $\mathsf{JS} \cup \{\mathsf{u}\}$ system.

Definition 1.9. *The translation $[\![\cdot]\!]_{\mathsf{S}}$ from intuitionistic sequents into intuitionistic structures is defined recursively as follows:*

$$[\![\vdash A]\!]_{\mathsf{S}} = A \qquad \textit{and} \qquad [\![A, \Gamma \vdash B]\!]_{\mathsf{S}} = A \rightarrow [\![\Gamma \vdash B]\!]_{\mathsf{S}}$$

Now, we can prove soundness of the $\mathsf{JS} \cup \{\mathsf{u}\}$ system with respect to the sequent calculus for intuitionistic logic, and the proof is similar to the one used in the nested sequents systems of the previous chapter, so that we will not give all the details.

Remark 1.10. *Since the $\mathsf{JS} \cup \{\mathsf{u}\}$ system uses a congruence when building structures from formulas, we need to be sure that when the congruence rule $\equiv$ is used to rewrite the formula used to represent a structure into another formula, representing the same structure, this step can be performed in $\mathsf{LJ}_{\top} \cup \{\mathsf{cut}\}$. However, this is trivial, since the equations given in Figure 1 correspond to equivalences in intuitionistic logic.*

Theorem 1.11 (Soundness of $\mathsf{JS} \cup \{\mathsf{u}\}$)**.** *If some structure A is provable in $\mathsf{JS} \cup \{\mathsf{u}\}$, then the sequent $\vdash A$ is provable in the $\mathsf{LJ}_{\top} \cup \{\mathsf{cut}\}$ sequent calculus.*

Proof. We proceed by induction on the length of a proof $\mathscr{D}$ of A in $\mathsf{JS} \cup \{\mathsf{u}\}$. In the base case, when $\mathscr{D}$ is just a structure, it has to be the unit $\top$, and we are done since its translation $\top$ is provable in $\mathsf{LJ}_{\top} \cup \{\mathsf{cut}\}$ by using the $\top_{\mathsf{R}}$ rule. In the general case, we consider the bottommost instance r in $\mathscr{D}$:

$$\mathsf{r}\,\frac{\begin{array}{c}\Vdots\\ \xi\{C\}\end{array}}{\xi\{B\}}$$

We have to show first that the implication $C \to B$ is provable in $\mathsf{LJ}_\top \cup \{\mathsf{cut}\}$, and for this we use a case analysis on r and build a proof Π_1 of this implication. In each case, we can exhibit a proof of this implication. Then, given some context ξ, we can prove by a straightforward induction on ξ, and by invertibility of the $\to_\mathsf{R}$ rule, that there is a proof Π_2 of $\xi\{C\} \vdash \xi\{B\}$ in $\mathsf{LJ}_\top \cup \{\mathsf{cut}\}$. By induction hypothesis, we also have a proof Π_3 of $\xi\{C\}$, and therefore we can build a proof of $\xi\{B\}$ by using a cut as follows:

$$\mathsf{cut}\ \dfrac{\overset{\Pi_3}{\vdash \xi\{C\}} \qquad \overset{\Pi_2}{\xi\{C\} \vdash \xi\{B\}}}{\vdash \xi\{B\}}$$

□

Then, the proof of completeness provides a translation in the other direction, and is also similar to the one used in the nested sequents.

Theorem 1.12 (Completeness of $\mathsf{JS} \cup \{\mathsf{u}\}$)**.** *If some sequent $\Gamma \vdash A$ is provable in the $\mathsf{LJ}_\top \cup \{\mathsf{cut}\}$ sequent calculus, then the structure $[\![\Gamma \vdash A]\!]_\mathsf{S}$ is provable in $\mathsf{JS} \cup \{\mathsf{u}\}$.*

Proof. By induction on a proof Π of the sequent $\Gamma \vdash A$ in $\mathsf{LJ}_\top \cup \{\mathsf{cut}\}$, and by using a case analysis on the bottommost rule instance r in Π, we build a proof $\mathscr{D}$ of the translation of this sequent in the $\mathsf{JS} \cup \{\mathsf{u}\}$ system. This is straightforward in every cases, and it can require to compose the proofs obtained by induction on different branches, as in the case of the cut — where the derivation $\mathscr{D}'_1$ is obtained by plugging $\mathscr{D}_1$ inside a context, and for a $\Sigma = C_1, \cdots, C_n$ we denote by $\Sigma \to D$ the structure $C_1 \to \cdots \to C_n \to D$ here:

$$\mathsf{cut}\ \dfrac{\overset{\Pi_1}{\Delta \vdash B} \qquad \overset{\Pi_2}{\Psi, B \vdash A}}{\Delta, \Psi \vdash A} \quad \longrightarrow \quad \mathsf{u}\ \dfrac{\mathsf{s}^*\ \dfrac{\Psi \to ((\Delta \to B) \to B) \to A}{\Delta \to \Psi \to (B \to B) \to A}}{\Delta \to \Psi \to A} \qquad \begin{array}{c} \mathscr{D}_2 \\ \Psi \to B \to A \\ \mathscr{D}'_1 \end{array}$$

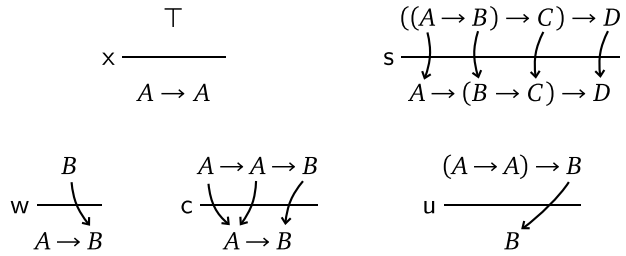
which is possible because of the deep inference methodology. Notice that the rule $\to_\mathsf{R}$ and the two unit rules $\top_\mathsf{L}$ and $\top_\mathsf{R}$ are transparent in this translation, and the rule $\to_\mathsf{L}$ is handled by simply composing the proofs corresponding to the branches, without adding any rule instance. □

Now that we know that this calculus of structures is a valid proof system for intuitionistic logic, we can turn to the dynamics of the proofs, and we shall adapt the procedure used to eliminate cuts in nested sequents to this setting.

1.3 Cut Elimination and Normalisation

The system $\mathsf{JS} \cup \{\mathsf{u}\}$ contains the cut rule u, which allows to introduce new atoms in a structure, and it should be eliminated to get a normal form. Although this system behaves in many ways like the nested sequents systems of the previous chapter, the cut elimination argument is a little simpler, due to the absence of equivalent of the left and right implication rules — it is however almost the same proof.

Because of the similarity between the $\mathsf{JS} \cup \{\mathsf{u}\}$ system and the nested sequents systems defined in the previous chapters, we can transfer the concepts defined in this setting into the calculus of structures. This includes of course the definition of the multiplicity, but also the notions of ancestor and the scope of a rule instance. In the rules of the $\mathsf{JS} \cup \{\mathsf{u}\}$ system, the connections inducing the flow-graph are the following — where the connections not shown in the switch rule are as expected:



Our goal here is to adapt the technique of cut permutations, in order to translate any proof in $\mathsf{JS} \cup \{\mathsf{u}\}$ into a proof of the same structure in the *cut-free* system JS, where the u rule is not available. The main difference with the setting of nested sequents is that formulas introduced by a cut can be immediately used, while with sequents they must be orderly decomposed into the meta-level first. An immediate consequence is that given a proof in $\mathsf{JS} \cup \{\mathsf{u}\}$, its cut instances can be decomposed into atomic cuts without looking for the matching left or right implication rules, so that we can assume that given a proof in $\mathsf{JS} \cup \{\mathsf{u}\}$, any cut instance is of the shape:

$$\mathsf{u}\,\frac{\xi\{(a \to a) \to B\}}{\xi\{B\}}$$

The proof of this property relies on the adaptation of the merging lemma to this setting, allowing to rewrite a proof according to the result of the cut decomposition.

Lemma 1.13 (Merging)**.** *For any proof $\mathscr{D}$ of $\xi\{(A \to \zeta\{\top\}^+ \to B) \to C\}$ in $\mathsf{JS} \cup \{\mathsf{u}\}$, there is also a proof of $\xi\{(\zeta\{A\}^+ \to B) \to C\}$ in this system.*

Proof. We consider the particle $\underline{\kappa}$ of the structure $(A \to \zeta\{\top\}^+ \to B)$ used in the conclusion of the proof $\mathscr{D}$, and proceed by induction on the pair $(\mathscr{M}_{\mathscr{D}}(\underline{\kappa}), \mathscr{H}(\mathscr{D}))$, under lexicographic order. At each step we use a case analysis on the bottommost instance r in $\mathscr{D}$, to rewrite this instance into a derivation of the desired conclusion.

1. If r affects only A, we use the induction hypothesis on the proof $\mathscr{D}_1$ above r and then use the result to build a new proof, with at the bottom the instance r used on A once inside each copy in ζ:

$$\mathsf{r}\,\frac{\xi\{(A' \to \zeta\{\top\}^+ \to B) \to C\}}{\xi\{(A \to \zeta\{\top\}^+ \to B) \to C\}} \quad\longrightarrow\quad \mathsf{r}^*\,\frac{\xi\{(\zeta\{A'\}^+ \to B) \to C\}}{\xi\{(\zeta\{A\}^+ \to B) \to C\}}$$

   In the special case where r is an axiom instance and deletes all of A, there is no need to use the induction hypothesis, we are done — since we start with a proof, which is a derivation with premise $\top$, this case must eventually occur.

2. If r does not affect A, we can proceed by induction hypothesis — this includes the case of a contraction or weakening inside ζ affecting some of the copies of $\top$, and this is the reason for the general statement of the theorem to involve an arbitrary number of holes in ζ.

3. If r is a switch moving another structure D to the left of A, we can apply the induction hypothesis to the proof $\mathscr{D}_1$ above r and use several contractions and switches at the bottom of the resulting proof:

$$\mathsf{s}\,\frac{\xi\{((D \to A) \to \zeta\{\top\}^+ \to B) \to C\}}{\xi\{D \to (A \to \zeta\{\top\}^+ \to B) \to C\}} \longrightarrow \begin{array}{c} \mathsf{s}^* \dfrac{\xi\{(\zeta\{D \to A\}^+ \to B) \to C\}}{\xi\{D \to \cdots \to D \to (\zeta\{A\}^+ \to B) \to C\}} \\ \mathsf{c}^* \dfrac{\phantom{\xi\{D \to \cdots \to D \to (\zeta\{A\}^+ \to B) \to C\}}}{\xi\{D \to (\zeta\{A\}^+ \to B) \to C\}} \end{array}$$

4. If r is a weakening which deletes the structure $(A \to \zeta\{\top\}^+ \to B)$, then we can simply rewrite the conclusive structure, and there is no need to apply the induction hypothesis, we are done.

5. If r is a contraction which duplicates the structure $(A \to \zeta\{\top\}^+ \to B)$, we can rewrite the conclusive structure and also the premise, and apply the induction hypothesis twice — this is possible, since the multiplicity of $\underline{\kappa}$ has decreased.

In any case, we have rewritten the proof so that in its conclusion, the structure A appears inside the holes of $\zeta\{\ \}$ rather than outside of this context, so that we have exactly the expected proof, obtained by transformation of the original one. □

Now, we can show how to reduce a cut instance to the atomic form in a given proof in $\mathsf{JS} \cup \{\mathsf{u}\}$, using a global rewriting of the whole part of the proof located above this cut — notice that this cannot work in general for derivations.

Proposition 1.14. *Given a proof in* $\mathsf{JS} \cup \{\mathsf{u}\}$ *using non-atomic cuts, there is a proof of the same conclusion in* $\mathsf{JS} \cup \{\mathsf{u}\}$ *using only* u *instances in the atomic form.*

Proof. By induction on the number of non-atomic cut instances in the given proof $\mathscr{D}$, we show that they can be decomposed into atomic instances. In the base case, there are only atomic cuts in $\mathscr{D}$ and we are done. In the general case, we pick some non-atomic cut instance r, introducing a non-atomic structure that must be of the shape $C \to D$, as follows:

$$\mathsf{u}\,\frac{\overset{\mathscr{D}_1}{\Vdots}\ \xi\{((C \to D) \to (C \to D)) \to B\}}{\xi\{B\}}$$

and we can use an induction on the size of the structure $C \to D$, decomposing at each step the cut into two cuts on the smaller structures C and D, as follows:

$$\mathsf{u}\,\frac{\mathsf{u}\,\dfrac{\overset{\mathscr{D}'_1}{\Vdots}\ \xi\{((((C \to C) \to D) \to D)) \to B\}}{\xi\{(D \to D) \to B\}}}{\xi\{B\}}$$

where the proof $\mathscr{D}_1'$ is obtained by applying Lemma 1.13 to the proof $\mathscr{D}_1$, to move the structure C inside the negative structure $C \to D$ and thus make it match the result of using twice the cut rule. When we reach the base case of this induction, we have replaced the initial cut with several atomic cuts, and we can go on with the main induction hypothesis, to decompose the other cut instances. □

This result will greatly simplify the proof of cut elimination, since we can thus restrict permutations to atomic cut instances, which have only limited interaction with other instances. Indeed, the only *matching* instances are those affecting one of the atoms introduced by such a cut. However, there is still the problem that switch instances might be blocked above a cut that we need to permute upwards. To solve this problem, we define one synthetic cut rule, which associates several switches to the cut, as follows — where $\Delta \to F$ is a short notation for a structure of the shape $C_1 \to \cdots \to C_n \to F$, here:

$$\mathsf{us}\ \frac{((\Delta \to A) \to A) \to B}{\Delta \to B}$$

and we can simply eliminate atomic instances of this rule, which is reduced to the standard cut in the case where Δ is actually only the $\top$ unit.

Theorem 1.15 (Cut elimination)**.** *Any proof $\mathscr{D}$ of a structure A in $\mathsf{JS} \cup \{\mathsf{u}\}$ can be transformed into a cut-free proof $\mathscr{D}'$ of A in JS.*

Proof. If there is no cut instance in the given proof $\mathscr{D}$, we are done. In the general case, we proceed by induction on the number of cut instances in $\mathscr{D}$, and consider the topmost cut instance r in $\mathscr{D}$, with the proof $\mathscr{D}_2$ above it. Moreover, because of Proposition 1.14, we can assume without loss of generality that this cut is atomic. Then, we proceed by induction on the pair $(\mathscr{M}_{\mathscr{D}}(\mathsf{r}), \mathscr{H}(\mathscr{D}_2))$ to show that it can be eliminated, using a case analysis on the bottommost instance r_1 in $\mathscr{D}_2$:

$$\mathsf{r}\ \dfrac{\mathsf{r}_1\ \dfrac{\mathscr{D}_2\Big\Vert\ \xi\{D\}}{\xi\{((\Delta \to a) \to a) \to B\}}}{\xi\{\Delta \to B\}}$$

1. If r_1 is not in the scope of r, we can permute it down and proceed by induction hypothesis, because the height of $\mathscr{D}_2$ has decreased.
2. If r_1 is a switch moving a structure to the left of the positive a introduced by the cut, we can assimilate it into the cut us instance, and then we can use the induction hypothesis, since the height of $\mathscr{D}_2$ has decreased.
3. If r_1 is an axiom instance matching r, it can interact with r and disappear, and we apply the main induction hypothesis, since one cut was eliminated by the following transformation:

$$\mathsf{us}\ \dfrac{\mathsf{x}\ \dfrac{\xi\{a \to B\}}{\xi\{((a \to a) \to a) \to B\}}}{\xi\{a \to B\}} \quad \longrightarrow \quad \xi\{a \to B\}$$

4. If r_1 is a weakening erasing the principal structure of r, we simply remove the cut and keep the weakening, and then go on by induction hypothesis, since one cut was erased by this transformation:

$$\dfrac{\dfrac{\xi\{B\}}{\xi\{((\Delta \to a) \to a) \to B\}}\,\mathsf{w}}{\xi\{\Delta \to B\}}\,\mathsf{us} \quad\longrightarrow\quad \dfrac{\xi\{B\}}{\xi\{\Delta \to B\}}\,\mathsf{w}^*$$

5. If r_1 is a contraction duplicating the principal structure of r, we duplicate the cut and compose the two copies, so that from the configuration:

$$\dfrac{\dfrac{\xi\{((\Delta \to a) \to a) \to ((\Delta \to a) \to a) \to B\}}{\xi\{((\Delta \to a) \to a) \to B\}}\,\mathsf{c}}{\xi\{\Delta \to B\}}\,\mathsf{us}$$

we obtain the following replacement derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{((\Delta \to a) \to a) \to ((\Delta \to a) \to a) \to B\}}{\xi\{((\Delta \to a) \to a) \to \Delta \to B\}}\,\mathsf{us}}{\xi\{\Delta \to \Delta \to B\}}\,\mathsf{c}^*}{\xi\{\Delta \to B\}}\,\mathsf{us}$$

We can use the induction hypothesis on the topmost copy of the cut, because its multiplicity is smaller than the multiplicity of the original instance. Then, it is also possible to apply the induction hypothesis on the resulting proof glued above the bottommost copy, because the cut elimination process cannot increase the multiplicity of the bottommost copy, so that it is at most one less than the original multiplicity. Finally, we use the main induction hypothesis, since one cut was eliminated.

This means that to eliminate all cut instances in the given proof, we simply have to repeatedly apply the reduction steps described above, and we will reach the point where no cut is left — as mentioned, if the given proof uses non-atomic cut, we have to rewrite the proof into a new, similar one, where all cuts are atomic. □

This cut elimination procedure suffers from the same problem as the procedure devised in the setting of nested sequents, in the sense that given any arbitrary proof, which possibly uses non-atomic cut instances, it requires some global rewriting of the proof to produce a result, because of the merging lemma, which is necessary to decompose cuts to the atomic form. As in the previous chapter, a solution is to introduce the dual of the switch rule, and to perform a *normalisation* on the whole dual *up fragment* — that is, to eliminate all instances of the rules dual to the basic rules of the JS system — which will be local. Indeed, the rewriting described in the merging lemma is essentially the same in JS as in JN, and the solution based on the dual of the switch is also valid here.

$$\mathsf{x}{\downarrow}\ \frac{\top}{A \to A} \qquad \mathsf{c}{\downarrow}\ \frac{A \to A \to B}{A \to B}$$

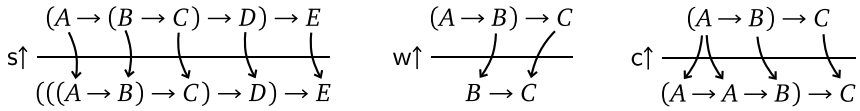
$$\mathsf{s}{\downarrow}\ \frac{((A \to B) \to C) \to D}{A \to (B \to C) \to D} \qquad \mathsf{w}{\downarrow}\ \frac{B}{A \to B}$$

$$\mathsf{s}{\uparrow}\ \frac{(A \to (B \to C) \to D) \to E}{(((A \to B) \to C) \to D) \to E} \qquad \mathsf{w}{\uparrow}\ \frac{(A \to B) \to C}{B \to C}$$

$$\mathsf{x}{\uparrow}\ \frac{(A \to A) \to B}{\top \to B} \qquad \mathsf{c}{\uparrow}\ \frac{(A \to B) \to C}{(A \to A \to B) \to C}$$

Figure 3: Inference rules for the system SJS

Symmetric normalisation. The set of inference rules for the symmetric system SJS is given in Figure 3, and it is defined for structures using the same grammar and the same equations as in $\mathsf{JS} \cup \{\mathsf{u}\}$. The down fragment, where rules have names of the form $\mathsf{r}{\downarrow}$, corresponds to the cut-free system JS, and the cut rule u has been renamed $\mathsf{x}{\uparrow}$ to remain consistent with its duality to the axiom rule $\mathsf{x}{\downarrow}$. The up fragment is composed of the dual of the inference rules in the down fragment, and the first step is again to specify which connexions are induced by the new inference rules in the flow-graph of a proof in SJS.



Notice that once again, not all connections are shown, but the connections not written, in the switch rules, follow as expected the indexes on structures, and thus connect in particular the implications where A appears on the left. This yields the notion of flow-graph for proofs in the SJS system, and we can therefore transpose all definitions concerning $\mathsf{JS} \cup \{\mathsf{u}\}$ into this extended setting. Then, once again, we can restrict our study to the cases where all instances of $\mathsf{x}{\uparrow}$ are atomic.

Proposition 1.16. *Given a proof in* SJS *using non-atomic instances of the* $\mathsf{x}{\uparrow}$ *rule, there is a proof of the same conclusion in* SJS *using only atomic* $\mathsf{x}{\uparrow}$ *instances.*

Proof. By induction on the number of non-atomic cut instances in the given proof $\mathscr{D}$, we show that they can be replaced with a derivation of SJS using cuts only in the atomic form. In the base case, there are only atomic cuts in $\mathscr{D}$ and we are done.

In the general case, we pick a non-atomic cut in $\mathscr{D}$, introducing a non-atomic structure which must be of the shape $C \to D$, and we use an induction on the size of the structure $C \to D$, decomposing at each step the cut in two cuts on the smaller structures C and D, by replacing it with the derivation shown on the right below:

$$\mathsf{x}\uparrow \dfrac{\xi\{((C \to D) \to (C \to D)) \to B\}}{\xi\{B\}} \quad \longrightarrow \quad \mathsf{x}\uparrow \dfrac{\mathsf{s}\uparrow \dfrac{\xi\{(C \to (C \to D) \to D) \to B\}}{\xi\{(((C \to C) \to D) \to D) \to B\}}}{\xi\{B\}}$$

□

As in the case of nested sequents, for the local normalisation proof we need to use a synthetic form of the identity and cut and rules, incorporating switches. From the basic versions, we generalise the $\mathsf{x}\downarrow$ and $\mathsf{x}\uparrow$ rules into the following rules:

$$\mathsf{xs}\downarrow \dfrac{\Delta}{(\Delta \to A) \to A} \qquad \mathsf{xs}\uparrow \dfrac{((\Delta \to A) \to A) \to B}{\Delta \to B}$$

where $\Delta \to F$ denotes as usual a structure of the shape $C_1 \to \cdots \to C_n \to F$. Notice that the $\mathsf{xs}\downarrow$ rule cannot be applied at toplevel — outside of any context — if Δ is not reduced to one structure C. We now define the measure we need in the proof, based on the one from the nested sequents setting. Note that the definition for the context weight of a $\mathsf{s}\uparrow$ instance can be imported immediately in this setting.

Definition 1.17. *In a derivation $\mathscr{D}$ in* SJS*, the* rank *of an instance* r *of any up rule, denoted by $\mathscr{R}_{\mathscr{D}}(\mathsf{r})$, is the pair $(\mathscr{M}_{\mathscr{D}}(\mathsf{r}), |\mathsf{r}|)$, under lexicographic ordering, where $|\mathsf{r}|$ is* 0 *for any rule except for* $\mathsf{s}\uparrow$*, for which it is the context weight of the instance.*

Finally, we can prove the normalisation result, which defines the local rewriting procedure to eliminate all up rule instances from a given proof in SJS.

Theorem 1.18 (Local normalisation)**.** *Any proof $\mathscr{D}$ of a structure A in* SJS *can be transformed into a proof $\mathscr{D}'$ of A in* SJ*.*

Proof. If there is no up rule instance in the given proof $\mathscr{D}$, we are done, since $\mathscr{D}$ is a valid proof in the down fragment SJN. In the general case, we proceed by induction on the number of up rule instances in $\mathscr{D}$, and consider the topmost such instance r in $\mathscr{D}$, with the proof $\mathscr{D}_2$ above it. Moreover, because of Proposition 1.16, we can assume without loss of generality that all $\mathsf{x}\uparrow$ instances in $\mathscr{D}$ are atomic. Then, we proceed by induction on the pair $(\mathscr{R}_{\mathscr{D}}(\mathsf{r}), \mathscr{H}(\mathscr{D}_2))$ to show that such an up instance can be eliminated, using a case analysis on the bottommost instance r_1 in $\mathscr{D}_2$:

$$\mathsf{r} \dfrac{\mathsf{r}_1 \dfrac{\overset{\mathscr{D}_2}{\xi\{D\}}}{\xi\{((C \to a) \to a) \to B\}}}{\xi\{C \to B\}} \qquad \textit{or} \qquad \mathsf{r}\uparrow \dfrac{\mathsf{r}_1 \dfrac{\overset{\mathscr{D}_2}{\xi\{G\}}}{\xi\{F\}}}{\xi\{E\}}$$

1. If r_1 is not in the scope of r, we can permute it down and proceed by induction hypothesis, because the height of $\mathscr{D}_2$ has decreased.

2. If r_1 is a switch moving a structure to the left of the positive a introduced by the cut, we can assimilate it into the cut $\mathsf{x}\uparrow$ instance, and then we can use the induction hypothesis, since the height of $\mathscr{D}_2$ has decreased.

3. If r_1 is an axiom instance matching r, it can interact with r and disappear, and we apply the main induction hypothesis, since one cut was eliminated by the following transformation, in the case of an axiom on the positive a:

$$\dfrac{\dfrac{\xi\{a \to B\}}{\xi\{((a \to a) \to a) \to B\}}\,\mathsf{x}\downarrow}{\xi\{a \to B\}}\,\mathsf{x}\uparrow \quad \longrightarrow \quad \xi\{a \to B\}$$

and the following transformation, in the other case:

$$\dfrac{\dfrac{\xi\{C \to a\}}{\xi\{((C \to a) \to a) \to a\}}\,\mathsf{x}\downarrow}{\xi\{C \to a\}}\,\mathsf{x}\uparrow \quad \longrightarrow \quad \xi\{C \to a\}$$

4. If r_1 is a weakening erasing the principal structure of r, we simply remove the cut and keep the weakening, and then go on by induction hypothesis, since one cut was erased by this transformation:

$$\dfrac{\dfrac{\xi\{B\}}{\xi\{((C \to a) \to a) \to B\}}\,\mathsf{w}\downarrow}{\xi\{C \to B\}}\,\mathsf{x}\uparrow \quad \longrightarrow \quad \mathsf{w}\downarrow\dfrac{\xi\{B\}}{\xi\{C \to B\}}$$

5. If r_1 is a contraction duplicating the principal structure of r, we duplicate the cut and compose the two copies, so that from the configuration:

$$\dfrac{\dfrac{\xi\{((C \to a) \to a) \to ((C \to a) \to a) \to B\}}{\xi\{((C \to a) \to a) \to B\}}\,\mathsf{x}\uparrow}{\xi\{C \to B\}}\,\mathsf{c}\downarrow$$

we obtain the following replacement derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{((C \to a) \to a) \to ((C \to a) \to a) \to B\}}{\xi\{((C \to a) \to a) \to C \to B\}}\,\mathsf{x}\uparrow}{\xi\{C \to C \to B\}}\,\mathsf{c}\downarrow}{\xi\{C \to B\}}$$

We can use the induction hypothesis on the topmost copy of the cut, because its multiplicity is smaller than the multiplicity of the original instance. Then, it is also possible to apply the induction hypothesis on the resulting proof glued above the bottommost copy, because the cut elimination process cannot increase the multiplicity of the bottommost copy, so that it is at most one less than the original multiplicity. Finally, we use the main induction hypothesis, since one cut was eliminated.

The remaining cases are the new cases introduced by the use of the symmetric system, with a dual up fragment, and in particular the dual switch rule s which is used to decompose cuts to the atomic form.

Given any up rule instance r, the transformation to perform, when the instance r_1 above is a weakening or a contraction erasing or duplicating the whole structure that was changed by r, is always the same. In such a case, we treat r the same way as we treated the situation of a cut used below a structural rule:

6. If r_1 is a weakening w↓ instance erasing the whole structure affected by r, we just have to erase r, and we can go on using the main induction hypothesis, since one up rule instance was eliminated:

$$\mathsf{w}\downarrow\dfrac{\dfrac{\xi\{G\}}{\xi\{F\to G\}}}{\mathsf{r}\dfrac{}{\xi\{E\to G\}}}\quad\longrightarrow\quad\mathsf{w}\downarrow\dfrac{\xi\{G\}}{\xi\{E\to G\}}$$

7. If r_1 is a contraction c↓ instance duplicating the structure affected by r, then we can duplicate r and go on by induction hypothesis, which is possible since after this transformation, r has a smaller multiplicity than the original:

$$\mathsf{c}\downarrow\dfrac{\dfrac{\xi\{F\to F\to G\}}{\xi\{F\to G\}}}{\mathsf{r}\dfrac{}{\xi\{E\to G\}}}\quad\longrightarrow\quad\mathsf{r}\dfrac{\dfrac{\xi\{F\to F\to G\}}{\xi\{E\to F\to G\}}}{\mathsf{r}\dfrac{}{\mathsf{c}\downarrow\dfrac{\xi\{E\to E\to G\}}{\xi\{E\to G\}}}}$$

In other cases, either the permutation is a trivial one, or we provide a rewriting to apply in this situation. We start with the elimination of dual switch s↑ instances, which requires the introduction of the other up rules w↑ and c↑:

8. If r_1 is an axiom x↓ instance matching the s↑ instance r, we integrate the dual switch inside the axiom, and go on using the main induction hypothesis since we have removed one up rule instance:

$$\mathsf{x}\downarrow\dfrac{\xi\{(B\to D)\to E\}}{\mathsf{s}\uparrow\dfrac{\xi\{(B\to(C\to C)\to D)\to E\}}{\xi\{(((B\to C)\to C)\to D)\to E\}}}\quad\longrightarrow\quad\mathsf{x}\downarrow\dfrac{\xi\{(B\to D)\to E\}}{\xi\{(((B\to C)\to C)\to D)\to E\}}$$

9. If r_1 is a weakening w↓ instance matching the dual switch, we have to turn the s↑ instance into a dual weakening w↑ instance, shown below, and we can go on by induction hypothesis, since the new w↑ instance has exactly the same multiplicity as the original r instance, but the height of $\mathscr{D}_2$ has decreased:

$$\mathsf{w}\downarrow\dfrac{\xi\{(B\to D\to E)\to F\}}{\mathsf{s}\uparrow\dfrac{\xi\{(((B\to C)\to D)\to E)\to F\}}{\xi\{((((B\to C)\to D)\to E)\to F\}}}\quad\longrightarrow\quad\mathsf{w}\downarrow\dfrac{\mathsf{w}\uparrow\dfrac{\xi\{(B\to D\to E)\to F\}}{\xi\{(D\to E)\to F\}}}{\xi\{((((B\to C)\to D)\to E)\to F\}}$$

10. If r_1 is a contraction $\mathsf{c}{\downarrow}$ instance matching the dual switch, the $\mathsf{s}{\uparrow}$ instance is duplicated when it permutes with the contraction, so that from the initial situation shown below:

$$\mathsf{c}{\downarrow}\dfrac{\mathsf{s}{\uparrow}\dfrac{\xi\{(B \to (C \to C \to D) \to E) \to F\}}{\xi\{(B \to (C \to D) \to E) \to F\}}}{\xi\{(((B \to C) \to D) \to E) \to F\}}$$

we obtain the new situation shown below, where a dual contraction instance $\mathsf{c}{\uparrow}$ has been created for glueing the proof above r_1 to the premise of the two dual switches, by collapsing the two copies of B created, after they have been moved outside of their immediate context by dual switches:

$$\mathsf{c}{\downarrow}\dfrac{\mathsf{s}{\uparrow}\dfrac{\mathsf{s}{\uparrow}\dfrac{\mathsf{c}{\uparrow}\dfrac{\xi\{(B \to (C \to C \to D) \to E) \to F\}}{\xi\{(B \to B \to (C \to C \to D) \to E) \to F\}}}{\xi\{(B \to (C \to (B \to C) \to D) \to E) \to F\}}}{\xi\{(((B \to C) \to (B \to C) \to D) \to E) \to F\}}}{\xi\{(((B \to C) \to D) \to E) \to F\}}$$

We can use the induction hypothesis on the dual contraction $\mathsf{c}{\uparrow}$ introduced, since its multiplicity is at most the same as the multiplicity of r — because B is in positive position. Then, we can also use the induction hypothesis on the two copies of the $\mathsf{s}{\uparrow}$ instance, since they have a smaller multiplicity than r as well, and finally we use the main induction hypothesis, because one up rule was eliminated.

11. If r_1 is a switch $\mathsf{s}{\downarrow}$ instance, there are two possible configurations in which the $\mathsf{s}{\uparrow}$ instance can interact with it, the first one happening when r_1 move a structure inside the structure moved by the $\mathsf{s}{\uparrow}$ instance, so that we have:

$$\mathsf{s}{\downarrow}\dfrac{\xi\{((A \to B) \to (C \to D) \to E) \to F\}}{\mathsf{s}{\uparrow}\dfrac{\xi\{A \to (B \to (C \to D) \to E) \to F\}}{\xi\{A \to ((B \to C) \to D) \to E) \to F\}}} \longrightarrow \mathsf{s}{\uparrow}\dfrac{\xi\{((A \to B) \to (C \to D) \to E) \to F\}}{\mathsf{s}{\downarrow}\dfrac{\xi\{((((A \to B) \to C) \to D) \to E) \to F\}}{\mathsf{s}{\downarrow}\dfrac{\xi\{((A \to (B \to C) \to D) \to E) \to F\}}{\xi\{A \to ((B \to C) \to D) \to E) \to F\}}}}$$

and we can use the induction hypothesis, since the height of $\mathscr{D}_2$ has decrased. The second situation is symmetric to the first one, so that we use the same rewriting but considered symmetrically, and then we can use the induction hypothesis on the topmost copy of $\mathsf{s}{\uparrow}$, since the height of $\mathscr{D}_2$ has decreased, and also on the copy below, since it has a smaller size than the original.

All other cases involving $\mathsf{s}{\uparrow}$ instances are trivial permutations. Then, we show how to treat the instances of the dual weakening $\mathsf{w}{\uparrow}$ rule, and this will never require to introduce instances of other rules than $\mathsf{w}{\uparrow}$ itself — as usual, permuting up a dual weakening erases some other rule instances.

12. If r_1 only affects a structure contained inside the structure introduced by r, it can simply be erased — this is dual to the situation where an up rule instance is erased by a weakening — and we can go on by induction hypothesis since the height of $\mathscr{D}_2$ has decreased.

13. If r_1 is an $\mathsf{x}{\downarrow}$ instance applied exactly on the structure introduced by r, the axiom can be erased, and we can go on by induction hypothesis, since one up rule instance was eliminated:

$$\dfrac{\dfrac{\xi\{(C \to D) \to E\}}{\xi\{(((C \to B) \to B) \to D) \to E\}}\,\mathsf{x}{\downarrow}}{\xi\{D \to E\}}\,\mathsf{w}{\uparrow} \quad \longrightarrow \quad \dfrac{\xi\{(C \to D) \to E\}}{\xi\{D \to E\}}\,\mathsf{w}{\uparrow}$$

14. If r_1 is an instance of $\mathsf{s}{\downarrow}$ moving material inside the structure introduced by r, we can erased this material and introduce it directly using a dual weakening, and then use the induction hypothesis, which is of course possible since the height of $\mathscr{D}_2$ has decreased:

$$\dfrac{\dfrac{\xi\{((C \to B) \to D) \to E\}}{\xi\{C \to (B \to D) \to E\}}\,\mathsf{s}{\downarrow}}{\xi\{C \to D \to E\}}\,\mathsf{w}{\uparrow} \quad \longrightarrow \quad \dfrac{\dfrac{\xi\{((C \to B) \to D) \to E\}}{\xi\{D \to E\}}\,\mathsf{w}{\uparrow}}{\xi\{C \to D \to E\}}\,\mathsf{w}{\downarrow}$$

All other cases involving $\mathsf{w}{\uparrow}$ instances are trivial permutations. Then, we show how to eliminate instances of the dual contraction $\mathsf{c}{\uparrow}$ rule, and this is similar to the situation described for dual weakenings:

15. If r_1 only affects a structure inside the structure resulting from the collapse of the two structures operated by the r instance, it can be duplicated — this is dual to the situation where an up instance is duplicated by a contraction — and we can use the induction hypothesis since the height of $\mathscr{D}_2$ has decreased.

16. If r_1 is an instance of $\mathsf{s}{\downarrow}$ moving material inside the structure resulting from the collapse operated by r, we use a contraction to duplicate the structure B and then use two copies of r_1 before we collapse the resulting structures with the dual contraction — and we can use the induction hypothesis, because the height of $\mathscr{D}_2$ has decreased:

$$\dfrac{\dfrac{\dfrac{\xi\{((B \to C) \to D) \to E\}}{\xi\{B \to (C \to D) \to E\}}\,\mathsf{s}{\downarrow}}{\xi\{B \to (C \to C \to D) \to E\}}\,\mathsf{c}{\uparrow} \quad \longrightarrow \quad \dfrac{\dfrac{\dfrac{\xi\{((B \to C) \to D) \to E\}}{\xi\{((B \to C) \to (B \to C) \to D) \to E\}}\,\mathsf{c}{\uparrow}}{\xi\{B \to (C \to (B \to C) \to D) \to E\}}\,\mathsf{s}{\downarrow}}{\xi\{B \to B \to (C \to C \to D) \to E\}}\,\mathsf{s}{\downarrow}}{\xi\{B \to (C \to C \to D) \to E\}}\,\mathsf{c}{\downarrow}$$

All other cases involving $\mathsf{c}{\uparrow}$ instances are trivial permutations. □

2 A System in Natural Deduction Style

The calculus of structures is naturally similar, in its design, to the sequent calculus, in the sense that it relies on dualities, and in particular on the duality between the identity rule and the cut rule. In almost all the systems that were designed in this setting, for intuitionistic as well as classical or linear logic, and other logics, there is some cut rule which can be shown admissible, one way or another. However, this is not the approach of *natural deduction*, which was introduced together with the sequent calculus [Gen34], as a formalism allowing to write formal proofs in a way that would reflect the way mathematicians write proofs in practice.

In such a proof system, logical connectives are not decomposed by left and right rules depending on their position in the initial formula, but rather *introduced* and *eliminated*. As a result, the left/right symmetry of the sequent calculus is lost, and the induced shape of proofs as well — for example, to prove some formula B under the hypothesis $A \to B$, it is impossible to decompose the implication on the left. In the setting of intuitionistic logic, one calculus of structures [BM08] was designed following this style, in order to extract an algorithmic interpretation, but the theory for this system remains largely undevelopped. We will present here another system based on the style of natural deduction, with the goal of establishing an adequate normalisation theory that can serve as basis for a computational interpretation.

We consider only the implication fragment of intuitionistic logic here, and we use the switch rule, which is an important and distinctive feature introduced by the deep inference methodology. Therefore, our system is significantly different from the existing calculus [BM08], which completely relies on a conjunction connective and has no switch rule. Our system could be considered as similar to the $\mathsf{JS} \cup \{\mathsf{u}\}$ system defined previously, but it uses the natural deduction style in the sense that it reintroduces a distinction between two levels in formulas, and it relies on rules for the introduction and elimination of the implication — these rules establish the meaning of the implication connective with respect to the meta-level implication, for which equations are used.

2.1 Basic Definitions

Once again, we assume given a countable set of *atoms*, which are denoted by small latin letters such as a, b, c, and the connective $\to$ for intuitionistic implication. We also use another implication connective $\Rightarrow$ which represents meta-level implication. Moreover, we need a unit, denoted by $\top$, which represents truth and will be the left unit of the meta-level implication.

Definition 2.1. *The* formulas *of intuitionistic logic are defined by the grammar:*

$$A, B \ ::= \ a \mid \top \mid A \to B \mid A \Rightarrow B$$

Then, the same equations as in JS are introduced on the meta-level implication, to keep the proof system simple to use. This forms the basis for the structures that will be manipulated — notice that there is no equation defined to manipulate the basic $\to$ implication connective, which cannot be directly handled.

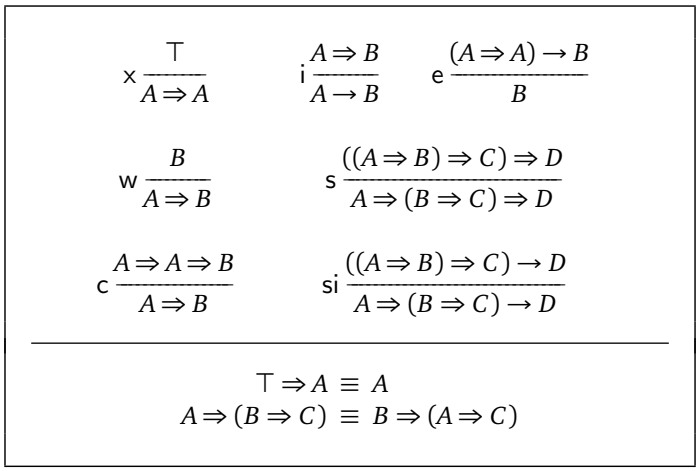


Figure 4: Inference rules and congruence for the system JD

Definition 2.2. *The* structures *of our system are defined as the equivalence classes of formulas generated by the congruence described by the equations shown in Figure 4.*

The idea is that the $\Rightarrow$ connective takes the role of the traditional $\vdash$ symbol in natural deduction, separating hypotheses from the goal formula, for which a proof should be built. Then, the definition of contexts is similar to the one used in JS, but a hole can be located only on the left of a meta-level implication $\Rightarrow$, so that we cannot manipulate the inside of structures at the basic level of the $\to$ connective, except under a meta-level implication inside it.

Definition 2.3. *The* contexts *of our system are structures with a hole* $\{\ \}$ *meant to be filled by another structure, and are defined by the following grammar:*

$$\xi \;::=\; \{\ \} \mid (\xi \Rightarrow A) \Rightarrow B \mid (\xi \Rightarrow A) \to B$$

Contexts will be denoted as $\xi\{\ \}$ or $\zeta\{\ \}$, so that for example $\xi\{A\}$ is the context ξ where the hole has been replaced by the structure A.

Inference rule instances are defined as usual, and the set of inference rules used in our natural deduction style system, that we will call JD, is given in Figure 4. The main difference with the $\mathsf{JS} \cup \{\mathsf{u}\}$ system presented previously is embodied in the introduction and elimination rules for $\to$ which are named i and e respectively. We have no proper cut rule in this setting, but the composition of an e instance and an i instance decomposing the implication involved in e is equivalent to the standard cut rule, so that we can define such a cut rule u as follows:

$$\mathsf{u}\,\frac{(A \Rightarrow A) \Rightarrow B}{B} \quad = \quad \mathsf{e}\,\frac{\mathsf{i}\,\dfrac{(A \Rightarrow A) \Rightarrow B}{(A \Rightarrow A) \to B}}{B}$$

Notice that the elimination rule e used in this system does not follow in any way the so-called *subformula property*, since it requires to choose a formula A, during proof construction, that should ideally help proving the goal formula B. Indeed, the process of proving a formula is different in this context, and in particular requires to expand the goal formula when the required hypothesis is not available.

Moreover, there are in this system two switch rules, namely s and si, because the decomposition of the cut into an introduction and an elimination rule creates the possibility to move material to the left of an implication before it is turned into a meta-level implication. We could choose to use only the si rule, while retaining completeness of the system, since this would simply force a certain organisation of switches in the proof — in particular, having a built-in switch in the elimination rule follows the idea of building the switch into the cut, and this would correspond to using only si instances right above an e instance. Without the si rule, the structure $A \to (A \to B) \to B$ would for example not be provable.

Example 2.4. *Because of the distinction kept between the basic level of formulas and the meta-level, there are less possibilities in the proof construction process in* JD *than in* JS. *Here is a proof of $A \to A$ based on the use of an elimination on A:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\top}{A \Rightarrow A}\,\mathsf{x}}{((A \Rightarrow A) \Rightarrow A) \Rightarrow A}\,\mathsf{x}}{(A \Rightarrow A) \Rightarrow A \Rightarrow A}\,\mathsf{s}}{(A \Rightarrow A) \to (A \to A)}\,\mathsf{i}^2}{A \to A}\,\mathsf{e}$$

As in any other proof system in the calculus of structures, we handle in a similar way proofs and open derivations, proofs being defined as derivations with the unit $\top$ for premise — and we will use the standard notations for derivations and proofs. There is however one important difference with standard calculi of structures: the axiom rule cannot easily be reduced to its atomic form. Indeed, it is not possible here to use a switch to move a structure to the left of a basic $\to$ implication, so that we cannot rewrite axiom instances on a structure of the shape $(A \to B) \Rightarrow (A \to B)$ into a proof using two axioms on A and B.

2.2 Correspondence to Natural Deduction

As usual, we will prove soundness and completeness for our system with respect to the corresponding traditional system, which is here the standard natural deduction system $\mathsf{NJ}_\top$ for intuitionistic logic. Its inference rules are shown in Figure 5, and it uses the truth unit denoted by $\top$. The translations between structures and sequents required to do this are straightforward, as the ones used in sequent style. Between structures and formulas, we use a simple translation in both directions.

Definition 2.5. *The intuitionistic formula/structure translation $\cdot^*$ from formulas to structures and from structures to formulas is defined as:*

$$a^* = a \qquad \top^* = \top \qquad (A \to B)^* = A^* \to B^* \qquad (A \Rightarrow B)^* = A^* \to B^*$$

$$\mathsf{ax}\ \frac{}{A \vdash A} \qquad \rightarrow_{\mathsf{e}} \frac{\Gamma \vdash A \quad \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash B} \qquad \mathsf{weak}\ \frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

$$\top\ \frac{}{\vdash \top} \qquad \rightarrow_{\mathsf{i}} \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \qquad \mathsf{cont}\ \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}$$

Figure 5: Inference rules for system $\mathsf{NJ}_\top$

Definition 2.6. *The translation $[\![\cdot]\!]_{\mathsf{S}}$ from intuitionistic sequents into intuitionistic structures in natural deduction style is defined recursively as follows:*

$$[\![\vdash A]\!]_{\mathsf{S}} = A^* \qquad \textit{and} \qquad [\![A, \Gamma \vdash B]\!]_{\mathsf{S}} = A^* \Rightarrow [\![\Gamma \vdash B]\!]_{\mathsf{S}}$$

Then, we can proof soundness of the JD proof system by translating any given proof of this system into a proof in the natural deduction system $\mathsf{NJ}_\top$. This relies on the use of many new instances of the $\rightarrow_{\mathsf{e}}$ elimination rule, which correspond to one part of the cuts introduced in the sequent calculus. Except for this decomposition of the cut rule, the proof of soundness of JD is the same as the one used to prove soundness of the $\mathsf{JS} \cup \{\mathsf{u}\}$ sequent calculus.

Theorem 2.7 (Soundness of JD)**.** *If some structure A is provable in the* JD *system, then the sequent $\vdash A^*$ is provable in the $\mathsf{NJ}_\top$ natural deduction system.*

Proof. We proceed by induction on the height of a proof $\mathscr{D}$ of A in JD. In the base case, when $\mathscr{D}$ is just a structure, it has to be the unit $\top$, and we are done since its translation $\top$ is provable in $\mathsf{NJ}_\top$ by using the $\top$ rule. In the general case, we consider the bottommost instance r in $\mathscr{D}$:

$$\mathsf{r}\ \frac{\begin{array}{c}\Vdots\\ \xi\{C\}\end{array}}{\xi\{B\}}$$

We have to show first that the implication $C^* \rightarrow B^*$ is provable in $\mathsf{NJ}_\top$, and for this we use a case analysis on r and build a proof Π_1 of this implication. In each case, we can exhibit a proof of this implication. Then, given some context ξ, we can prove by a straightforward induction on ξ that there is a proof Π_2 of the implication $\xi\{C\}^* \rightarrow \xi\{B\}^*$ in $\mathsf{NJ}_\top$. By induction hypothesis, we also have a proof Π_3 of $\xi\{C\}^*$, and therefore we can build a proof of $\xi\{B\}^*$ by using an elimination rule:

$$\rightarrow_{\mathsf{e}} \frac{\begin{array}{c}\Pi_3\\ \vdash \xi\{C\}^*\end{array} \qquad \begin{array}{c}\Pi_2\\ \vdash \xi\{C\}^* \rightarrow \xi\{B\}^*\end{array}}{\vdash \xi\{B\}^*}$$

□

Then, the proof of completeness provides a translation in the other direction, and is also similar to the one used in the nested sequents systems.

Theorem 2.8 (Completeness of JD). *If some sequent $\Gamma \vdash A$ is provable in the $\mathsf{NJ}_\top$ natural deduction system, then the structure $[\![\Gamma \vdash A]\!]_\mathsf{S}$ is provable in JD.*

Proof. By induction on a proof Π of the sequent $\Gamma \vdash A$ in $\mathsf{NJ}_\top$, and by using a case analysis on the bottommost rule instance r in Π, we build another proof $\mathscr{D}$ of the translation of this sequent in the JD system. This is straightforward in every cases, and can require to compose the proofs obtained by induction on different branches, as in the case of the elimination rule.

As an example, we show the case of the elimination rule, where the derivation $\mathscr{D}'_1$ is obtained by plugging $\mathscr{D}_1$ inside a context, and for a $\Sigma = C_1, \cdots, C_n$ we denote by $\Sigma \Rightarrow D^*$ the structure $C_1^* \Rightarrow \cdots \Rightarrow C_n^* \to D^*$ here:

$$\dfrac{\dfrac{\overset{\Pi_1}{\Delta \vdash B} \qquad \overset{\Pi_2}{\Psi \vdash B \to A}}{\Delta, \Psi \vdash A}\,{\to_\mathsf{e}}}{} \quad \longrightarrow \quad \dfrac{\dfrac{\dfrac{\overset{\mathscr{D}_2}{\Psi \Rightarrow B \to A}}{\Psi \Rightarrow ((\Delta \Rightarrow B) \Rightarrow B) \to A}\,{\mathsf{si}^*}}{\Delta \Rightarrow \Psi \Rightarrow (B \Rightarrow B) \to A}\,{\mathsf{e}}}{\Delta \Rightarrow \Psi \Rightarrow A}$$

which is possible because of the deep inference methodology used here, allowing to plug a derivation inside another one without fundamentally changing any of them. Notice that the translation of the $\top$ rule is transparent here, since the truth unit is the premise of a proof. □

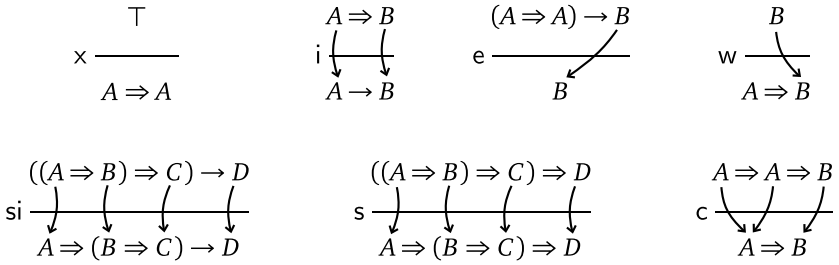
3 Detour Elimination

In natural deduction as well as in the JD proof system we have presented, there is no way of defining a normal form for proofs which would imply the completeness of a subsystem respecting the *subformula property*. There is however a phenomenon called *detour*, appearing when an introduction rule instance appears immediately above the corresponding elimination rule instance, as follows:

$$\dfrac{\overset{\Pi_1}{\Gamma \vdash A} \qquad \dfrac{\overset{\Pi_2}{\Delta, A \vdash B}}{\Delta \vdash A \to B}\,{\to_\mathsf{i}}}{\Gamma, \Delta \vdash B}\,{\to_\mathsf{e}} \qquad \text{or} \qquad \dfrac{\dfrac{\overset{\mathscr{D}}{\xi\{(A \Rightarrow A) \Rightarrow B\}}}{\xi\{(A \Rightarrow A) \to B\}}\,{\mathsf{i}}}{\xi\{B\}}\,{\mathsf{e}}$$

This situation corresponds to the use of a lemma A in the proof of the formula B, and can be reduced, in the natural deduction system $\mathsf{NJ}_\top$, by *inlining* the proof of this lemma: in the proof Π_2, all axioms using the A added to the goal formula on the right are replaced with the actual proof Π_1 of the formula A. In this standard setting, it is called *normalisation* [GLT89]. This can also be done in a decomposed way, by turning the detour $\to_\mathsf{e}/\to_\mathsf{i}$ into a cut instance, and then moving this cut upwards in the proof until it disappears — in the style of the sequent calculus.

This procedure, producing a proof in *normal form*, where no detour appears, from any given proof, can be adapted to the JD proof system. But in the calculus of structures, it is not easy to extract the branch Π_1 proving the formula A, so that the decomposed process, where a detour takes the form of a cut and is moved upwards one step at a time, is more natural. The proof of this ability to eliminate all detours in a given proof in JD thus follows the same scheme as the proof of cut elimination given in the setting of nested sequents, and for the $\mathsf{JS} \cup \{\mathsf{u}\}$ system. Once again, it is based on the definition of a toolset allowing to reason about the flow of particles in a derivation, adapted from the definitions given in Chapter 1. In this system, the connections inducing the flow-graph are similar to the ones of other systems:



Although the proof we are about to give is similar in principle to the one given for $\mathsf{JS} \cup \{\mathsf{u}\}$, there is a major technical difference, since the implication elimination rule, and thus also the cut rule u, cannot be reduced to its atomic form. Therefore, permutations are more complicated in this setting than it was in the sequent style system, because the use of complex cut structures implies that many inference rule instances might involve this structure and thus be blocked over the cut, so that we have to move all these rule instances along with a cut.

We will use a synthetic form of cut, corresponding to the use of a cut instance — which is itself the compound of an elimination e instance and an introduction i instance — below a number of other instances blocked above this cut. This new cut rule uc is defined below, where we use the antecedent notation Γ for a series of structures on the left of an implication, and the derivation $\mathscr{D}_1$ was plugged on top of the basic cut, all of its instances being in the scope of this u instance:

$$\mathsf{uc}\,\dfrac{C \Rightarrow B}{\Gamma \Rightarrow B} \quad = \quad \mathsf{u}\,\dfrac{\begin{array}{c} C \Rightarrow B \\ \mathscr{D}_1 \| \\ \Gamma \Rightarrow (A \Rightarrow A) \Rightarrow B \end{array}}{\Gamma \Rightarrow B}$$

For the sake of simplicity, we always consider all the elimination and introduction instances inside the cut as part of the proof — so that the cut is not really a proper inference rule, but rather a presentation. This allows us to consider the multiplicity $\mathscr{M}_{\mathscr{D}}(\mathsf{r})$ of some e instance r in a proof $\mathscr{D}$ even if it is seen inside some compound cut instance r′, since we have $\mathscr{M}_{\mathscr{D}}(\mathsf{r}) = \mathscr{M}_{\mathscr{D}}(\mathsf{r}')$ anyway. We also name the derivation $\mathscr{D}_1$ in a compound cut the *body* of this cut, and C is its *principal structure*.

There is however an intermediate step between the elimination of detours and the elimination of cut instances. Because of the interleaving of rule instances in a proof in the calculus of structures, the introduction instance of the i rule *matching* an elimination e instance — that is, introducing the implication eliminated by this i instance, or rather, the ancestor of the eliminated particle — is not necessarily located directly above this e instance in a given proof, so that elimination instances should be moved upwards as well, until they can be turned into a cut.

The first step of this process is to detect which e instances in a given proof $\mathscr{D}$ are involved in a detour, and these are exactly those that reach a matching introduction when moving upwards: if an e instance r eliminates some particle $\underline{\kappa}$ such that an ancestor $\underline{\upsilon}$ of $\underline{\kappa}$ is introduced by an i instance in $\mathscr{D}$, then r is said to be *involved in a detour*. This instance must then be permuted above other rule instances, and until it meets a matching introduction, it agglomerates instances blocked above, which is done through the definition of another synthetic rule, formed by an elimination and a derivation $\mathscr{D}_1$ of instances in the scope of this e instance:

$$\mathsf{ec}\,\dfrac{D \to B}{\Gamma \Rightarrow B} \quad = \quad \mathsf{e}\,\dfrac{\begin{array}{c} D \to B \\ \mathscr{D}_1 \| \\ \Gamma \Rightarrow (A \Rightarrow A) \to B \end{array}}{\Gamma \Rightarrow B}$$

Again, $\mathscr{D}_1$ is called the body of the rule, and D its principal structure. By considering rule instances appearing in a synthetic instance, we extend to the ec and uc rules the definitions given for other rules. In particular, the connexions they induce in the flow-graph of a proof are given by the connexions established through all the instances contained in such a synthetic instance.

We now have the tools to prove the result, stating that given an arbitrary proof in JD, we can build a proof $\mathscr{D}'$ with no detour in JD. This is done by introducing the synthetic rules ec and uc, into the system, as a notational artefact used to hide the interaction between elimination instances and the other instances blocked above.

Theorem 3.1 (Detour elimination)**.** *Any given proof $\mathscr{D}$ of a structure A in* JD *can be transformed into a proof $\mathscr{D}'$ of A with no detour in* JD*.*

Proof. If there is no detour in the proof $\mathscr{D}$, we are done. In the general case, we proceed by induction on the number of e instances involved in a detour in $\mathscr{D}$, and consider the topmost one in $\mathscr{D}$, that we see as a compound ec instance r, with the proof $\mathscr{D}_2$ above it. Then, we proceed by induction on the pair $(\mathscr{M}_{\mathscr{D}}(\mathsf{r}), \mathscr{H}(\mathscr{D}_2))$ to show that it can be eliminated, using a case analysis on the bottommost instance r_1 in $\mathscr{D}_2$, as described below:

$$\mathsf{r}\,\dfrac{\mathsf{r}_1\,\dfrac{\begin{array}{c} \mathscr{D}_2 \Vdots \\ \xi\{E\} \end{array}}{\xi\{D \to B\}}}{\xi\{\Gamma \Rightarrow B\}}$$

1. If r_1 is not in the scope of r, we can permute it down and proceed by induction hypothesis, because the height of $\mathscr{D}_2$ has decreased.

2. If r_1 affects only the principal structure of r, or if it is an si instance moving material from the context $\xi\{\cdot\}$ into D, we can assimilate it into the ec instance and go on by induction hypothesis, since $\mathscr{H}(\mathscr{D}_2)$ has decreased.

3. If r_1 is a matching introduction i instance, then we can assimilate it into r_1 to produce a cut u instance, which is seen as a compound uc instance, where $\mathscr{D}'_1$ is $\mathscr{D}_1$ with all instances of si replaced by s instances:

$$\dfrac{\dfrac{\dfrac{\xi\{D \Rightarrow B\}}{\xi\{D \to B\}}\,\mathsf{i}}{\vdots\,\mathscr{D}_1}}{\dfrac{\xi\{\Gamma \Rightarrow (A \Rightarrow A) \to B\}}{\xi\{\Gamma \Rightarrow B\}}\,\mathsf{e}} \quad\longrightarrow\quad \dfrac{\xi\{D \Rightarrow B\}}{\dfrac{\vdots\,\mathscr{D}'_1}{\dfrac{\xi\{\Gamma \Rightarrow (A \Rightarrow A) \Rightarrow B\}}{\xi\{\Gamma \Rightarrow B\}}\,\mathsf{u}}}$$

Notice that because of the implication connective eliminated by this e instance, no weakening or contraction instance can erase or duplicate the whole principal structure of r. If we reach the case where an i instance was encountered above the e instance to form a cut, we have to go on permuting this e instance up in the proof by moving the corresponding cut upwards. We also use a case analysis on the instance r_1 above the cut r, considered as a uc instance, in the following situation:

$$\dfrac{\dfrac{\vdots\,\mathscr{D}_2}{\dfrac{\xi\{E\}}{\xi\{C \Rightarrow B\}}\,\mathsf{r}_1}}{\xi\{\Gamma \Rightarrow B\}}\,\mathsf{r}$$

4. If r_1 is not in the scope of r, we can permute it down and proceed by induction hypothesis, because the height of $\mathscr{D}_2$ has decreased.

5. If r_1 is a switch s instance moving a structure inside C, we can assimilate it into the cut uc instance, and then we can use the induction hypothesis, since the height of $\mathscr{D}_2$ has decreased.

6. If r_1 is an axiom instance matching r, it can interact with r and disappear, and we apply the main induction hypothesis, since one cut was eliminated by the following transformation, in the case of an axiom on the positive A:

$$\dfrac{\dfrac{\xi\{A \Rightarrow B\}}{\xi\{((A \Rightarrow A) \Rightarrow A) \Rightarrow B\}}\,\mathsf{x}}{\xi\{A \Rightarrow B\}}\,\mathsf{uc} \quad\longrightarrow\quad \xi\{A \Rightarrow B\}$$

and by the following transformation, in the other case, where the body $\mathscr{D}_1$ of the cut is kept, since only the basic cut and axiom were unnecessary:

$$\dfrac{\dfrac{\xi\{C\}}{\xi\{(C \Rightarrow A) \Rightarrow A\}}\,\mathsf{x}}{\xi\{\Gamma \Rightarrow A\}}\,\mathsf{uc} \quad\longrightarrow\quad \dfrac{\xi\{C\}}{\vdots\,\mathscr{D}_1}\\ \xi\{\Gamma \Rightarrow A\}$$

7. If r_1 is a weakening erasing the principal structure of r, we simply remove the cut and replace the weakening with several weakenings, and then go on by induction hypothesis, since one cut was erased by this transformation:

$$\mathsf{uc}\,\dfrac{\mathsf{w}\,\dfrac{\xi\{B\}}{\xi\{(C \Rightarrow A) \Rightarrow B\}}}{\xi\{\Gamma \Rightarrow B\}} \quad\longrightarrow\quad \mathsf{w}^*\,\dfrac{\xi\{B\}}{\xi\{\Gamma \Rightarrow B\}}$$

8. If r_1 is a contraction duplicating the principal structure of r, we duplicate the cut and compose the two copies, so that from the configuration:

$$\mathsf{uc}\,\dfrac{\mathsf{c}\,\dfrac{\xi\{(C \Rightarrow A) \Rightarrow (C \Rightarrow A) \Rightarrow B\}}{\xi\{(C \Rightarrow A) \Rightarrow B\}}}{\xi\{\Gamma \Rightarrow B\}}$$

we obtain the following replacement derivation:

$$\mathsf{uc}\,\dfrac{\mathsf{uc}\,\dfrac{\mathsf{c}^*\,\dfrac{\xi\{(C \Rightarrow A) \Rightarrow (C \Rightarrow A) \Rightarrow B\}}{\xi\{\Gamma \Rightarrow \Gamma \Rightarrow B\}}}{\xi\{(C \Rightarrow A) \Rightarrow \Gamma \Rightarrow B\}}}{\xi\{\Gamma \Rightarrow B\}}$$

We can use the induction hypothesis on the topmost copy of the cut, because its multiplicity is smaller than the multiplicity of the original instance. Then, it is possible to apply the induction hypothesis on the resulting proof glued above the bottommost copy, because the detour elimination process cannot increase the multiplicity of the bottommost copy, so that it is at most one less than the original multiplicity. Finally, we use the main induction hypothesis, since one cut was eliminated. □

Variant systems. Just as is done in the nested sequents setting described in the previous chapter, different variants of the JD system can be defined based on other treatment of structural rules. The fully additive variant system JDa can be defined by building contraction and weakening, and the switch rules, inside other rules:

$$\mathsf{xw}\,\dfrac{\top}{\Delta \Rightarrow A \Rightarrow A} \qquad \mathsf{i}\,\dfrac{A \Rightarrow B}{A \to B} \qquad \mathsf{ea}\,\dfrac{\Delta \Rightarrow ((\Delta \Rightarrow A) \Rightarrow A) \to B}{\Delta \Rightarrow B}$$

where $\Delta \Rightarrow F$ is a short notation for a structure $C_1 \Rightarrow \cdots \Rightarrow C_n \Rightarrow F$, and here the rules should always be applied on a maximal substructure — in the sense that it is either at toplevel or directly on the left of another structure. Many other variants are possible, in particular using different versions of the elimination rule e, and of the switch rules. As in the nested sequents setting, all rewriting transformations defined on proofs in JD can be adapted, so that the detour elimination result can be proved, by a minimal adaptation of the proof given above, for JDa and the other variant systems. The interaction of a cut instance moving upwards with one of the » *compound* « rules can indeed be defined as a composition of the different interactions of a cut with the components of the rule.

Among other variants of JD, the one where only the switch rules are removed and si is built inside the e rule, that we call JDs, is interesting because it is rather close to the multiplicative presentation of intuitionistic logic in natural deduction. Also note that many variations are possible around the mechanism for contraction, as mentioned in Chapter 3. For example, one can use the rule:

$$\frac{(\Gamma \Rightarrow A) \Rightarrow (\Delta \Rightarrow A) \Rightarrow B}{(\Gamma \Rightarrow \Delta \Rightarrow A) \Rightarrow B}$$

where A is a plain intuitionistic formula — that is, a structure containing only basic implications $\rightarrow$ and no meta-level implications $\Rightarrow$. This rule performs a contraction but avoids the duplication of the whole substructure contained inside the structure being duplicated. One can simply add this rule to JDs to obtain JDr, where both kinds of contractions are available, and the detour elimination result can be proved in this mixed setting by following the scheme of the procedure that was described for JD above.

PART 3

Nested Proofs as Programs

Chapter 5

Nested Typing for Explicit Substitutions

This chapter presents an adaptation to the nested deduction setting of the basic principles of the typing methodology, as used for the Curry-Howard correspondence to provide a computational interpretation of proof systems in natural deduction and in the sequent calculus. In order to establish a connection to standard, well-known computational models, we explore here using this methodology the contents of the intuitionistic systems defined in Chapter 3 and Chapter 4. The result obtained is a variation of the standard type systems for λ-calculi with explicit substitutions and type systems for pure explicit substitutions, as presented in Chapter 2.

Although it might seem surprising that the computational contents of proofs in the deep inference setting can be described as λ-calculi, one should notice that this is not the only interpration that can be given for such proofs, and in particular this does not yield a perfect correspondence, as observed in NJ and the pure λ-calculus. The purpose of such a correspondence is to exhibit similarities between the shallow and the nested settings, so that particular features of nested deduction can be later pinpointed and incorporated into a well-understood framework. Indeed, in order to build an exact correspondence between proofs in nested deduction and a form of functional programs, these programs should be sequences of instructions, just as proofs are sequences of rule instances. But the sequence would not be *» executed «* one step after the other, but rather *reduced* with rewriting and interaction between instructions. This would induce a complex computational device — for this reason, we start here with the more familiar framework of functional programming.

There are two levels of variation on the standard, branching type systems. First, the systems in nested sequents are used as a new form of type system where typing judgements can appear within typing judgements, allowing to type λ-calculi with pure explicit substitutions. We study the properties of such type systems and show how cut elimination on the logical side is reflected as the transformation of typing derivation induced by reduction of terms in the calculus. Then, systems in natural deduction style in the calculus of structures are shown to yield a similar form of type systems, where typing judgements can also appear within types — not in types as obtained in the end for a given term, but during the typing process.

1 Typing with Nested Sequents

The standard way of exploring the computational contents of proofs in some given logical system, in the Curry-Howard tradition, is to use it as a type system for some computational language, such as the λ-calculus, or one of its variants [Gal93]. We show in this section how the nested sequents systems defined for intuitionistic logic in Chapter 3 can be viewed as type systems for λ-calculi described in Chapter 2, by establishing a correspondence between inference rules and typing rules. Because in this section, we are in the setting of nested sequents, we will be using only λ-calculi with pure explicit substitutions on the computational side.

1.1 Nested Typing Judgements

The starting point for the definition of a type system based on the nested sequents setting is the traditional notion of *typing judgements*, which expresses through the following syntax, similar to the syntax of sequents:

$$x_1 : B_1, \cdots, x_n : B_n \vdash t : A$$

that some given term t can be assigned type A under the assumption that each one of the x_i variables is assigned type B_i — the left part of the judgement, which holds these typing assumptions, is called the *typing environment* of the term t. But we are here in a nested setting, where a sequent can contain other sequents, and we need to change the notion of judgement accordingly. The benefit of keeping a sequent structure, with a *meta-level* in addition to the level of types, is that there is a clear distinction between the types assigned to different parts of a term, and the structure required to build the typing derivation. In this section, types are directly defined as intuitionistic formulas, without the truth unit, so that they can be a type variable a or a function type $A \to B$, for some types A and B.

Definition 1.1. *A* nested typing sequent δ *is a triple formed of a typing environment, a term and a type, generated through the following grammar:*

$$\delta ::= \Gamma \vdash t : A \qquad \Gamma ::= V_1, \cdots, V_n \qquad V ::= x : \{ \kappa \} \triangleright B \qquad \kappa ::= \delta_1 \parallel \cdots \parallel \delta_n$$

As we can see, a typing environment Γ is a set of *typing hypotheses*, where each hypothesis V_i is stating that some variable x has type B, under the condition that a set κ of other nested typing sequents can be validated. If κ is empty, we use the traditional writing $x : B$ to denote $x : \{\,\} \triangleright B$ and thus keep notations readable.

The typing process in this setting will follow the traditional scheme. Given some term t, we will build a *typing derivation* of the sequent $\vdash t : A$, for some type A. As done in the logical system, typing rules can be applied deeply inside nested typing sequents, and this requires to manipulate typing sequents with a hole.

Definition 1.2. *A typing* context *is a nested typing sequent with a hole* $\{\,\}$ *meant to be filled by another nested typing sequent, as defined by the following grammar:*

$$\xi ::= \{\,\} \mid \Gamma, x : \{ \kappa \parallel \xi \} \triangleright B \vdash t : A$$

$$\texttt{var}\ \frac{}{\Gamma, x : A \vdash x : A} \qquad \texttt{sub}\ \frac{\Gamma, x : \{\Gamma \vdash u : A\} \triangleright A \vdash t : B}{\Gamma \vdash t[x \leftarrow u] : B}$$

$$\texttt{lam}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \texttt{let}\ \frac{\Gamma, z : \{\kappa \parallel \Gamma \vdash t : A\} \triangleright B \vdash u : C}{\Gamma \ni x : \{\kappa\} \triangleright A \to B \vdash \texttt{let}\ z = x\ t\ \texttt{in}\ u : C}$$

Figure 1: Nested type system $\mathtt{N\overline{x}}$ for the $\lambda\overline{\mathtt{x}}$-calculus

1.2 Nested Typing for Pure Explicit Substitutions

The kind of terms that can be typed using a system based on nested typing sequents depends on its typing rules. We start our study with limited sets of rules, providing systems which can only work for basic λ-calculi of pure explicit substitutions. The use of more complicated rules for advanced operators is left to the next section.

The set of typing rules defining the nested type system $\mathtt{N\overline{x}}$ for the $\lambda\overline{\mathtt{x}}$-calculus is shown above in Figure 1, and these rules[1] can be applied inside any valid context. This is quite similar to standard typing rules for calculi of pure explicit substitutions, with the significant difference that we are using nesting instead of branching. The system has one typing rule for each construct of the language, and typing is there completely syntax-directed. These rules can be described as follows:

- the `var` rule says that some variable x has type A in any context Γ where the name x appears with this type.
- the `lam` rule says that an abstraction $\lambda x.t$ has type $A \to B$ in a context Γ if t has type B in the context Γ extended with the hypothesis that x has type A.
- the `let` rule says that the application of x to t in a term u is well-typed if we can type u with the additional hypothesis that the result z of the application has type B, under the condition that t has type A in the same context Γ used to type u — so that u is well-typed in a context where x has type $A \to B$.
- the `sub` rule says that a term t under a substitution $[x \leftarrow u]$ is well-typed if we can type t with the additional hypothesis that the variable x has type A, under the condition that u has type A in the same context used to type t.

Example 1.3. *Below is shown a simple typing derivation in the* $\mathtt{N\overline{x}}$ *system for some* $\lambda\overline{\mathtt{x}}$*-term* $\texttt{let}\ z = x\ y\ \texttt{in}\ z$ *under a context* Γ *containing typing assumptions on both of the free variables* y *and* x *in the term. Note that the last application of* `var` *cannot be moved down.*

$$\texttt{let}\ \frac{\texttt{var}\ \dfrac{\texttt{var}\ \dfrac{}{\Gamma, z : \{\emptyset\} \triangleright B \vdash z : B}}{\Gamma, z : \{\Gamma \ni y : A \vdash y : A\} \triangleright B \vdash z : B}}{\Gamma \ni x : A \to B, y : A \vdash \texttt{let}\ z = x\ y\ \texttt{in}\ z : B}$$

[1] Here we will write $\Gamma \ni V$ to denote that some typing assumption V appears in Γ.

$$\texttt{var}\ \frac{}{x:A \vdash x:A} \qquad \texttt{sub}\ \frac{\Gamma, \Delta, x:\{\Delta, \Psi \vdash u:A\} \triangleright A \vdash t:B}{\Gamma, \Delta, \Psi \vdash t[x \leftarrow u]:B}$$

$$\texttt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \texttt{rem}\ \frac{\Gamma \vdash t:B}{\Gamma, x:\{\kappa\} \triangleright A \vdash t:B}$$

$$\texttt{let}\ \frac{\Gamma, \Delta, z:\{\kappa \parallel \Delta, \Psi \vdash t:A\} \triangleright B \vdash u:C}{(\Gamma, \Delta, \Psi) \ni x:\{\kappa\} \triangleright A \to B \vdash \texttt{let}\ z = x\ t\ \texttt{in}\ u:C}$$

Figure 2: Nested type system $\mathtt{N}\overline{\mathtt{e}}$ for the $\lambda\overline{\mathtt{e}}$-calculus

This system can be refined to handle more complicated reduction behaviours, as can be found for example in the $\lambda\overline{\mathtt{e}}$-calculus. The fact that erasure and duplication of explicit substitutions are handled in a more subtle way is reflected by the use of a typing rule removing an hypothesis from a typing sequent, and in the modification of the `let` and `sub` typing rules to include both *copying* and *splitting* of hypotheses, which corresponds to the different cases of use of a variable in one or two subterms of an application or a substitution. Notice that at this point the language of terms remains the same as in $\lambda\overline{\mathtt{x}}$, but typing rules are no longer syntax-direct, since there is no explicit syntax in the $\lambda\overline{\mathtt{e}}$-calculus for the erasure of explicit substitutions.

The typing rules defining the $\mathtt{N}\overline{\mathtt{e}}$ system for $\lambda\overline{\mathtt{e}}$ are shown in Figure 2. The rule `rem` is introduced to handle erasure of hypotheses, and the `var` rule is modified, since there is no need anymore to erase the context — this can be done by using the `rem` rule, although erasure need not be done only below an axiom. Then, the `sub` and `let` rules are also modified to handle the distribution of hypotheses among typing sequents.

Example 1.4. *Below is shown an example of a typing derivation in the* $\mathtt{N}\overline{\mathtt{e}}$ *system, similar to the example given for* $\mathtt{N}\overline{\mathtt{x}}$ *but where the more refined distribution of typing hypotheses is illustrated.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{var}\ \dfrac{}{\Gamma, z:\{\emptyset\} \triangleright B \vdash z:B}}{\texttt{rem}\ \dfrac{}{w:\{\Gamma \vdash u:A\} \triangleright B, z:\{\emptyset\} \triangleright B \vdash z:B}}}{\texttt{var}\ \dfrac{}{w:\{\Gamma \vdash u:A\} \triangleright B, z:\{y:A \vdash y:A\} \triangleright B \vdash z:B}}}{\texttt{rem}\ \dfrac{}{x:A \to B, w:\{\Gamma \vdash u:A\} \triangleright B, z:\{y:A \vdash y:A\} \triangleright B \vdash z:B}}}{\texttt{let}\ \dfrac{}{x:A \to B, y:A, w:\{\Gamma \vdash u:A\} \triangleright B \vdash \texttt{let}\ z = x\ y\ \texttt{in}\ z:B}}}{\texttt{let}\ \dfrac{}{\Gamma, x:A \to B, y:A \vdash \texttt{let}\ w = x\ u\ \texttt{in}\ \texttt{let}\ z = x\ y\ \texttt{in}\ z:B}}$$

It also shows how unused assumptions are erased: the hypothesis on the type of x *is no longer used after the treatment of the two applications, and it has be erased with the* `rem` *rule. Notice that the erasure of the assumption on* w *implies the erasure of the whole condition attached to it, with its own set* Γ *of assumptions.*

$$\texttt{var}\ \frac{}{x:A \vdash x:A} \qquad \texttt{sub}\ \frac{\Gamma, x:\{\,\Delta \vdash u:A\,\} \triangleright A \vdash t:B}{\Gamma, \Delta \vdash t[x \leftarrow u]:B}$$

$$\texttt{rem}\ \frac{\Gamma \vdash t:B}{\Gamma, x:\{\,\kappa\,\} \triangleright A \vdash t:B} \qquad \texttt{dup}\ \frac{\Gamma, x:\{\,\kappa\,\} \triangleright A, y:\{\,\kappa\,\} \triangleright A \vdash t_{[y/x]}:B}{\Gamma, x:\{\,\kappa\,\} \triangleright A \vdash t:B}$$

$$\texttt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \texttt{let}\ \frac{\Gamma, z:\{\,\kappa \parallel \Delta \vdash t:A\,\} \triangleright B \vdash u:C}{\Gamma, \Delta, x:\{\,\kappa\,\} \triangleright A \to B \vdash \texttt{let}\ z = x\ t\ \texttt{in}\ u:C}$$

Figure 3: Nested type system $\mathtt{N\overline{s}}$ for the $\lambda\overline{\mathtt{s}}$-calculus

This type system suffers from the same problem as the $\lambda\overline{\mathtt{e}}$-calculus itself, having an easy treatment of erasure, but a complicated shape for rules including potential duplication. This is fixed by introducing a rule for duplication, which simplifies the `sub` and `let` rules, with the price that this rule is not syntax-directed, since there is also no explicit syntax for duplication in the corresponding calculus $\lambda\overline{\mathtt{s}}$. The typing rule `dup` and the other modified rules are shown below in Figure 3. The fact that duplication of hypotheses is handled separatly simplifies the scheme of the `sub` and `let` rules. This type system for $\lambda\overline{\mathtt{s}}$ is called $\mathtt{N\overline{s}}$, by the same scheme as above.

Remark 1.5. *In the* `dup` *rule, there must be at least two occurrences of the variable x in the term t. Otherwise, its renamed version $t_{[y/x]}$ would be exactly the same as t, modulo renaming, and the rule could induce infinite loops during typing.*

Example 1.6. *Below is shown an example of a typing derivation in the $\mathtt{N\overline{s}}$ system, for a $\lambda\overline{\mathtt{s}}$-term corresponding to the application of a function x to twice the same argument a, that would be written $x\ a\ a$ in the standard λ-calculus.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{w:\{\emptyset\} \triangleright B \vdash w:B}\,\texttt{var}}{w:\{\,b:A \vdash b:A\,\} \triangleright B \vdash w:B}\,\texttt{var}}{w:\{\,a:A \vdash a:A \parallel b:A \vdash b:A\,\} \triangleright B \vdash w:B}\,\texttt{var}}{a:A, z:\{\,b:A \vdash b:A\,\} \triangleright A \to B \vdash \texttt{let}\ w = z\ a\ \texttt{in}\ w:B}\,\texttt{let}}{x:A \to A \to B, a:A, b:A \vdash \texttt{let}\ z = x\ b\ \texttt{in let}\ w = z\ a\ \texttt{in}\ w:B}\,\texttt{let}}{x:A \to A \to B, a:A \vdash \texttt{let}\ z = x\ a\ \texttt{in let}\ w = z\ a\ \texttt{in}\ w:B}\,\texttt{dup}$$

This illustrates the use of a separate rule for duplication, that must be used whenever a typing assumption, such as the one on a here, is required to type two different parts of a term. The renaming involved leads us to replace an occurrence of a by the fresh variable b in this example, so that one variable in the initial term will match the copy of the assumption created by applying this rule.

1.3 Properties of Nested Typing with Sequents

Although it is a variant of the standard typing mechanism used for λ-calculi, nested typing has particular properties, due to the structure of the rules it uses. Indeed, we cannot establish a perfect correspondence between the tree-based syntactic shape of λ-terms and the sequential structure of nested typing derivations. Therefore, we have a mismatch, which causes a term, in $\lambda\overline{x}$ and other pure explicit substitutions calculi, to have several typing derivations[2].

However, we can prove that nested type systems have enough good properties to be used on the pure explicit substitutions calculi presented in Chapter 2. We now concentrate on the properties of the $\mathtt{N}\overline{x}$ system for the $\lambda\overline{x}$-calculus, which is the basis for other, more refined calculi, and show how results extend to these. First, a crucial result is that the computation of the type to be assigned to a given $\lambda\overline{x}$-term is a terminating process, so that given a term, the typing procedure either returns a type, or it fails if no rule can be applied while the derivation is not complete. We achieve this through a measure, defined at first on terms.

Definition 1.7. *The* weight *of a $\lambda\overline{x}$-term t is defined recursively as:*

$$\begin{array}{rlrl} \mathtt{W}(x) &= 1 & \mathtt{W}(\mathtt{let}\ z = x\ v\ \mathtt{in}\ u) &= 1 + 2 \times \mathtt{W}(u) + \mathtt{W}(u) \times \mathtt{W}(v) \\ \mathtt{W}(\lambda x.u) &= 2 \times \mathtt{W}(u) + 1 & \mathtt{W}(u[x \leftarrow v]) &= 1 + \mathtt{W}(u) + \mathtt{W}(u) \times \mathtt{W}(v) \end{array}$$

Then, the notion of weight is extended to handle nested typing judgements, by assigning a value to a $\lambda\overline{x}$-term under a certain typing environment. Because of the recursive nature of nested typing judgements, this measure is defined on different kind of objects, involved in the definition, and we overload notations for simplicity.

Definition 1.8. *Given a $\lambda\overline{x}$-term t and a typing environment Γ, the* weight *of t under the environment Γ is denoted by $\mathtt{W}(\Gamma, t)$ and defined as $\mathtt{W}(\Gamma) \times \mathtt{W}(t)$, where the weight $\mathtt{W}(\Gamma)$ of the environment Γ is defined recursively as:*

$$\mathtt{W}(\emptyset) = 1 \qquad \mathtt{W}(x : \{\kappa\} \triangleright A) = \mathtt{W}(\kappa) \qquad \mathtt{W}(x : \{\kappa\} \triangleright A, \Delta) = \mathtt{W}(\kappa) + \mathtt{W}(\Delta)$$

and the weight $\mathtt{W}(\kappa)$ of a sequence κ of typing judgements as:

$$\mathtt{W}(\emptyset) = 1 \qquad \mathtt{W}(\Delta \vdash u : A) = \mathtt{W}(\Delta, u) \qquad \mathtt{W}(\Delta \vdash u : A \parallel \kappa) = \mathtt{W}(\Delta, u) + \mathtt{W}(\kappa)$$

This is all we need to prove that typing in the $\mathtt{N}\overline{x}$ system is terminating, which means that this computation will always provide a result, disregarding the choices made of which typing rule to apply at any point during the process.

Theorem 1.9. *The typing process in $\mathtt{N}\overline{x}$ is terminating.*

Proof. For some $\lambda\overline{x}$-term t under a typing environment Γ, we proceed by induction on $\mathtt{W}(\Gamma, t)$. In the base case, we can only apply the $\mathtt{var}$ rule on t, which is a variable x, and typing immediately terminates. In the general case, we consider any term u under an environment Δ to which we can apply a typing rule — if there is none, then typing fails and thus terminates immediately. Notice that this u can be either t, or any other term to be typed inside a condition in Γ.

[2] This is not the case in the traditional sequent calculus setting, as we have seen in Chapter 2, since exchange of left implication instances corresponds to the exchange of applications, but the branching structure is here flattened into a sequence.

Once this term u is chosen, we use a case analysis on all the typing rules that can be applied to it, most of them providing as premise a term v under an environment Φ, such that v is either t or a subterm of t, and Φ is a modification of Γ. Therefore, from t under Γ we obtain r under an environment Σ, by replacing u by v and Δ by Φ in t and Γ. In each case, we show that we have $\mathtt{W}(\Sigma, r) < \mathtt{W}(\Gamma, t)$, as follows:

1. If u is a variable x, we apply the `var` rule which has no premise, so that this term x appearing in a condition is removed from Γ and thus $\mathtt{W}(\Sigma) < \mathtt{W}(\Gamma)$.
2. If u of the shape $\lambda x.p$, then we have $\mathtt{W}(\Phi) = \mathtt{W}(\Delta)+1$ but $\mathtt{W}(u) < 2\times\mathtt{W}(v)+1$, so that after application of the `lam` rule and replacement we obtain a term r of less weight under its environment Σ than $\mathtt{W}(\Gamma, t)$, because $\mathtt{W}(\Delta) \geq 1$.
3. If u is of the shape $\mathtt{let}\ z = x\ q\ \mathtt{in}\ p$, then Δ has the shape $\Omega, x : \{\kappa\} \triangleright A \to B$, so that $\mathtt{W}(\Delta, u) = (\mathtt{W}(\Omega)+\mathtt{W}(\kappa)) \times (1+2\times\mathtt{W}(p)+\mathtt{W}(p)\times\mathtt{W}(q))$ and by the `let` rule we obtain $\mathtt{W}(\Phi, v) = (\mathtt{W}(\Omega)+\mathtt{W}(\kappa)+\mathtt{W}(\kappa)+(\mathtt{W}(\Omega)+\mathtt{W}(\kappa))\times\mathtt{W}(q))\times\mathtt{W}(p)$, which is less.
4. If u is of the shape $p[x \leftarrow q]$, then $\mathtt{W}(\Delta, u) = \mathtt{W}(\Delta)\times(1+\mathtt{W}(p)+\mathtt{W}(p)\times\mathtt{W}(q))$, and by the `sub` rule, we obtain less, $\mathtt{W}(\Phi, v) = (\mathtt{W}(\Delta)+\mathtt{W}(\Delta)\times\mathtt{W}(q))\times\mathtt{W}(p)$.

We conclude that, whatever typing rule we use, and term u we pick, the measure $\mathtt{W}(\Gamma, t)$ decreases by application of this rule. □

Remark 1.10. *At each step in the typing process, for a term and a typing environment, there is only finitely many possibilities of applying a typing rule. The typing procedure being terminating, there are finitely many typing derivations that can be built for some term t under an environment Γ, but in general more than one.*

It can also be shown that typing in $\mathtt{N}\overline{\mathtt{x}}$ is consistent, in the sense that to a given $\lambda\overline{\mathtt{x}}$-term it can only assign one unique type — up to renaming. This holds for the same reasons as in standard systems[3].

Proposition 1.11. *For any typing environment Γ and any $\lambda\overline{\mathtt{x}}$-term t, there is at most one type A such that $\Gamma \vdash t : A$ in $\mathtt{N}\overline{\mathtt{x}}$, up to renaming of base types in A.*

The extension of these properties to other calculi is simply a matter of ensuring that the new typing rules, as well as the modifications of basic rules, cannot break the given proofs in an essential way. The case of the $\mathtt{N}\overline{\mathtt{e}}$ type system for $\lambda\overline{\mathtt{e}}$ is easy, we can use the same measure as for $\mathtt{N}\overline{\mathtt{x}}$ — the `rem` rule is never a problem. But the case of the $\mathtt{N}\mathtt{s}$ system for $\lambda\mathtt{s}$ is slightly more complex, because of the `dup` rule, which is not completely syntax-directed. The only modification of the term t is the renaming of some x into y, and thus we need to extend the induction measure for termination, with the multiset of the number of occurrences of each variable in t — either free or bound, this is stable under α-conversion. Notice that termination is easy to obtain in such basic systems, but it does not hold in general, for any type system closer to the actual implementations of function programming languages.

[3]The nested type systems we provided here are all described through the same kind of inductive rules as standard systems: termination might have been a question because of nested, and the possible duplication of typing obligation, but consistency is not a problem.

2 Cut Elimination as Reduction

Now that we have established the static properties of nested typing, when used on λ-calculi with pure explicit substitutions, we can turn to the dynamic properties of such systems. Indeed, the main purpose of typing is to ensure the good behaviour of chosen terms during reduction, since a type system implicitly defines a subset of terms for which we usually have properties such as normalisation.

The logical foundation for the operational behaviour of typed terms is formed by the cut elimination result that was obtained on the side of logic, within the proof system — in an internal way, as done in Chapter 3 on systems for intuitionistic logic. The crucial observation is the correspondence between the reduction rules of the calculus and the steps used in the cut elimination procedure, each of them pushing a cut upwards in the proof. We conclude that the properties of the cut elimination procedure can be adapted as properties of typed terms, through the corresponding typing derivations.

We will now show the correspondence between the reduction rules of the basic $\lambda\overline{\mathrm{x}}$-calculus and cut elimination in the $\mathsf{JNa} \cup \{\mathsf{esa}\}$ proof system, and then discuss the properties we deduce from this. After this, we will see how this correspondence extends to other, more refined λ-calculi and proof systems. As we have seen, there is no exact correspondence between terms and typing derivations in systems based on nested sequents, — all systems shown previously suffer from this mismatch. But the relation between cut elimination steps and reduction rules is enough to prove that reduction can be safely performed in typed terms.

2.1 Reduction in the $\lambda\overline{\mathrm{x}}$-calculus

The $\lambda\overline{\mathrm{x}}$-calculus is the most basic calculus with pure explicit substitutions we have presented. Its reduction system $\longrightarrow_{\lambda\overline{\mathrm{x}}}$ is quite reduced and treats duplications and erasures of substitutions in the naïve way. This behaviour corresponds to a purely additive treatment of rules, as can be observed in the $\mathsf{JNa} \cup \{\mathsf{esa}\}$ proof system for intuitionistic logic. We establish the correspondence between the two by unfolding all cases and showing how they relate.

The reduction cases are mostly just local transformations, and we show how to rewrite a part of the typing derivation the same way a subderivation was rewritten in the cut elimination procedure. In the complicated cases, that involve a non-local rewriting, we explain the effect of the transformation on the term typed.

1. The reduction rule $t[x \leftarrow y] \longrightarrow_{\mathtt{ren}} t\{y/x\}$ corresponds to the interaction between a cut and an identity instance:

$$\dfrac{\dfrac{}{\xi\{\Gamma, x : A \vdash t : B\}}\,\mathtt{var}}{\xi\{\Gamma, x : \{\Gamma \vdash y : A\} \triangleright A \vdash t : B\}}\,\mathtt{sub} \quad\longrightarrow\quad \xi\{\Gamma \ni y : A \vdash t\{y/x\} : B\}$$

   in this simple situation, in most cases. This transformation is local concerning typing rules, but it requires to rename x into y in the rest of the derivation, above this part, to match the judgement using $y : A$ as hypothesis.

In another situation, the `ren` reduction rule corresponds to a more complex rewriting, because the cut and the identity instance are not located one above the other, so that the derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\Delta, w : \{\,\kappa\,\} \triangleright E \vdash v : B\}}{\xi\{\Delta, w : \{\,\Gamma \vdash y : C \to D \parallel \kappa\,\} \triangleright E \vdash v : B\}}\,\mathtt{var} \quad \overset{\mathscr{D}}{\|} \quad \xi\{\Gamma, z : \{\,\Gamma \vdash y : C \to D \parallel \Gamma, x : \{\,\Gamma \vdash y : C \to D\,\} \triangleright C \to D \vdash u : C\,\} \triangleright D \vdash t : B\}}{\xi\{\Gamma, x : \{\,\Gamma \vdash y : C \to D\,\} \triangleright C \to D \vdash \mathtt{let}\ z = x\ u\ \mathtt{in}\ t : B\}}\,\mathtt{let}}{\xi\{\Gamma \ni y : C \to D \vdash (\mathtt{let}\ z = x\ u\ \mathtt{in}\ t)[x \leftarrow y] : B\}}\,\mathtt{sub}$$

is turned into a simpler derivation, where $\mathscr{D}'$ is obtained from $\mathscr{D}$ by removal of a structure inside a deep context, that was not modified by $\mathscr{D}$, as follows:

$$\dfrac{\dfrac{\xi\{\Delta, w : \{\,\kappa'\,\} \triangleright E \vdash v\{y/x\} : B\}}{\xi\{\Gamma, z : \{\,\Gamma \vdash u\{y/x\} : C\,\} \triangleright D \vdash t\{y/x\} : B\}}\,\mathtt{let}}{\xi\{\Gamma \ni y : C \to D \vdash \mathtt{let}\ z = y\ u\{y/x\}\ \mathtt{in}\ t\{y/x\} : B\}}$$

2. The reduction rule $x[x \leftarrow u] \longrightarrow_{\mathtt{var}} u$ corresponds to the other possible form of interaction between a cut instance and an identity instance, where these instances are removed, and $\mathscr{D}'$ is obtained by extracting $\mathscr{D}$ from its context:

$$\dfrac{\dfrac{\dfrac{\xi\{\,\}}{\xi\{\Gamma, x : A \vdash x : A\}}\,\mathtt{var} \quad \overset{\mathscr{D}}{\|} \quad \xi\{\Gamma, x : \{\,\Gamma \vdash u : A\,\} \triangleright A \vdash x : A\}}{\xi\{\Gamma \vdash x[x \leftarrow u] : A\}}\,\mathtt{sub} \quad \longrightarrow \quad \dfrac{\xi\{\,\}}{\xi\{\Gamma \vdash u : A\}}$$

3. The reduction rule $z[x \leftarrow u] \longrightarrow_{\mathtt{nov}} z$ corresponds to the erasure of the whole cut when it encounters an implicit weakening, as follows:

$$\dfrac{\dfrac{\dfrac{\xi\{\,\}}{\xi\{\Gamma, x : \{\,\Delta \vdash v : C\,\} \triangleright A \vdash z : B\}}\,\mathtt{var} \quad \overset{\mathscr{D}}{\|} \quad \xi\{\Gamma, x : \{\,\Gamma \vdash u : A\,\} \triangleright A \vdash z : B\}}{\xi\{\Gamma \ni z : B \vdash z[x \leftarrow u] : B\}}\,\mathtt{sub} \quad \longrightarrow \quad \dfrac{\xi\{\,\}}{\xi\{\Gamma \ni z : B \vdash z : B\}}\,\mathtt{var}$$

4. The reduction rule $(\lambda y.t)[x \leftarrow u] \longrightarrow_{\mathtt{lam}} \lambda y.t[x \leftarrow u]$ corresponds to the permutation of a cut over an introduction rule, so that from the derivation:

$$\dfrac{\dfrac{\xi\{\Gamma, x : \{\,\Gamma \vdash u : A\,\} \triangleright A, y : B \vdash t : C\}}{\xi\{\Gamma, x : \{\,\Gamma \vdash u : A\,\} \triangleright A \vdash \lambda y.t : B \to C\}}\,\mathtt{lam}}{\xi\{\Gamma \vdash (\lambda y.t)[x \leftarrow u] : B \to C\}}\,\mathtt{sub}$$

we obtain the following new derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{lam}\,\dfrac{\xi\{\Gamma, y:B, x:\{\Gamma, y:C \vdash u:A\} \triangleright A \vdash t:C\}}{\xi\{\Gamma, y:B \vdash t[x \leftarrow u]:C\}}}{\xi\{\Gamma \vdash \lambda y.t[x \leftarrow u]:B \to C\}}$$

5. The rule $(\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u] \longrightarrow_{\mathtt{let}} \mathtt{let}\ y = z\ v[x \leftarrow u]\ \mathtt{in}\ t[x \leftarrow u]$ corresponds to the permutation of a cut over an application when it does not involve the main formula of the cut, so that from the derivation:

$$\mathtt{let}\,\dfrac{\mathtt{sub}\,\dfrac{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A, y:\{\kappa \parallel \Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash v:B\} \triangleright C \vdash t:D\}}{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash \mathtt{let}\ y = z\ v\ \mathtt{in}\ t:D\}}}{\xi\{\Gamma \ni z:\{\kappa\} \triangleright B \to C \vdash (\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u]:D\}}$$

we obtain the following new derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{sub}\,\dfrac{\mathtt{let}\,\dfrac{\xi\{\Gamma, y:\{\iota\} \triangleright C, x:\{\Gamma, y:\{\iota\} \triangleright C \vdash u:A\} \triangleright A \vdash t:D\}}{\xi\{\Gamma, y:\{\kappa \parallel \Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash v:B\} \triangleright C \vdash t[x \leftarrow u]:D\}}}{\xi\{\Gamma, y:\{\kappa \parallel \Gamma \vdash v[x \leftarrow u]:B\} \triangleright C \vdash t[x \leftarrow u]:D\}}}{\xi\{\Gamma \ni z:\{\kappa\} \triangleright B \to C \vdash \mathtt{let}\ y = z\ v[x \leftarrow u]\ \mathtt{in}\ t[x \leftarrow u]:D\}}$$

where $\iota = \kappa \parallel \Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash v:B$.

6. The rule $(\mathtt{let}\ y = x\ v\ \mathtt{in}\ t)[x \leftarrow \lambda z.u] \longrightarrow_{\mathtt{apd}} t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u]$ corresponds to the interaction between the cut and an application when the main formula of the cut is the one involved in the application, so that from the following derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{let}\,\dfrac{\mathtt{lam}\,\dfrac{\xi\{\Sigma, y:\{\Gamma, z:A \vdash u:B \parallel \Sigma \vdash v:A\} \triangleright B \vdash t:C\}}{\xi\{\Sigma, y:\{\Gamma \vdash \lambda z.u:A \to B \parallel \Sigma \vdash v:A\} \triangleright B \vdash t:C\}}}{\xi\{\Gamma, x:\{\Gamma \vdash \lambda z.u:A \to B\} \triangleright A \to B \vdash \mathtt{let}\ y = x\ v\ \mathtt{in}\ t:C\}}}{\xi\{\Gamma \vdash (\mathtt{let}\ y = x\ v\ \mathtt{in}\ t)[x \leftarrow \lambda z.u]:C\}}$$

where $\Sigma = \Gamma, x:\{\Gamma \vdash \lambda z.u:A \to B\} \triangleright A \to B$, we obtain the derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{sub}\,\dfrac{\xi\{\Sigma, y:\{\Sigma, z:\{\Sigma \vdash v:A\} \triangleright A \vdash u:B\} \triangleright B \vdash t:C\}}{\xi\{\Sigma, y:\{\Sigma \vdash u[z \leftarrow v]:B\} \triangleright B \vdash t:C\}}}{\xi\{\Gamma, x:\{\Gamma \vdash \lambda z.u:A \to B\} \triangleright A \to B \vdash t[y \leftarrow u[z \leftarrow v]]:C\}}}{\xi\{\Gamma \vdash t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u]:C\}}$$

This rewriting of the typing derivation is slightly more complicated than the others, since it is non-local, and this requires to adapt the derivation above the part transformed in order to move the instances used to type the term v inside the context where u is typed. However, there is no need here to deal with the distribution of typing assumptions, because of the additive setting, so that this rewriting does not require the introduction of new instances.

7. The reduction rule $t[x \leftarrow \mathtt{let}\ y = z\ v\ \mathtt{in}\ u] \longrightarrow_{\mathtt{out}} \mathtt{let}\ y = z\ v\ \mathtt{in}\ t[x \leftarrow u]$ corresponds to the permutation of a cut over an application when it affects the inside of the cut formula, so that from the derivation:

$$\mathtt{let}\ \frac{\mathtt{sub}\ \dfrac{\xi\{\Gamma, x : \{\Gamma, y : \{\kappa \parallel \Gamma \vdash v : B\} \triangleright C \vdash u : A\} \triangleright A \vdash t : D\}}{\xi\{\Gamma, x : \{\Gamma \vdash \mathtt{let}\ y = z\ v\ \mathtt{in}\ u : A\} \triangleright A \vdash t : D\}}}{\xi\{\Gamma \ni z : \{\kappa\} \triangleright B \to C \vdash t[x \leftarrow \mathtt{let}\ y = z\ v\ \mathtt{in}\ u] : D\}}$$

we obtain the following new derivation:

$$\mathtt{sub}\ \frac{\xi\{\Gamma, y : \{\kappa \parallel \Gamma \vdash v : B\} \triangleright C, x : \{\Gamma, y : \{\kappa \parallel \Gamma \vdash v : B\} \triangleright C \vdash u : A\} \triangleright A \vdash t : D\}}{\mathtt{let}\ \dfrac{\xi\{\Gamma, y : \{\kappa \parallel \Gamma \vdash v : B\} \triangleright C \vdash t[x \leftarrow u] : D\}}{\xi\{\Gamma \ni z : \{\kappa\} \triangleright B \to C \vdash \mathtt{let}\ y = z\ v\ \mathtt{in}\ t[x \leftarrow u] : D\}}}$$

Notice that the correspondence between steps of cut elimination and reduction rules of $\lambda\overline{\mathtt{x}}$ is highly similar to the correspondence between cut elimination in the shallow setting of the sequent calculus, and reduction rules of $\lambda\mathtt{x}$, apart from the replacement of branching with nesting. In this set of reductions, the `out` rule is a little different from the others: the reduction shown on the typing derivation is valid in general, but this rewriting is applied only in the particular case where the term t under the substitution on x is an application on x. This restriction is needed to preserve confluence, as explained in Chapter 2, and the fact that a more general reduction preserves typing guarantees that this rule is logically valid. All the cases listed above lead to the following expected result.

Proposition 2.1 (Subject reduction in $\lambda\overline{\mathtt{x}}$)**.** *If* $\Gamma \vdash t : A$ *and* $t \longrightarrow_{\lambda\overline{\mathtt{x}}} u$ *then* $\Gamma \vdash u : A$.

The observation that the typing of a term is preserved by reduction is the crucial step in the correspondence between cut elimination and reduction in the calculus, and intuitively states that all reduction rules in $\lambda\overline{\mathtt{x}}$ are logically valid. One can then prove the progress property, stating that a well-typed term is either a normal form, or can be reduced. This is a consequence of the correspondence between the typing derivations of $\mathtt{N}\overline{\mathtt{x}}$ and proofs in $\mathsf{JNa} \cup \{\mathsf{esa}\}$, as explicit substitutions are typed by cuts. These properties are the reason why the following theorem holds, extracting from typing the existence of a normalising strategy for a given term.

Theorem 2.2. *Given a* $\lambda\overline{\mathtt{x}}$*-term* t *well-typed in* $\mathtt{N}\overline{\mathtt{x}}$*, there exists a terminating strategy that allows to compute the strong normal form of* t *for* $\longrightarrow_{\lambda\overline{\mathtt{x}}}$.

Proof. By detour elimination in $\mathsf{JNa} \cup \{\mathsf{esa}\}$, since it was proved that all cuts can be eliminated from a given proof in this system. Through the correspondence shown above, all redexes can be eliminated from the term t. □

2.2 Reduction in Other Calculi

In more refined λ-calculi, the correspondence between cut elimination in a logical system and reduction rules is established the same way as in $\lambda\overline{\mathtt{x}}$, by refining some of the cases described above. We now consider the refinement to the $\lambda\overline{\mathtt{e}}$-calculus:

1. The reduction rule $t[x \leftarrow y] \longrightarrow_{\mathtt{ren}} t\{y/x\}$ corresponds to the interaction between a cut and an identity instance, in the simple case:

$$\mathtt{sub}\,\dfrac{\mathtt{var}\,\dfrac{\xi\{\Gamma, x : A \vdash t : B\}}{\xi\{\Gamma, x : \{\, y : A \vdash y : A\,\} \triangleright A \vdash t : B\}}}{\xi\{\Gamma, y : A \vdash t[x \leftarrow y] : B\}} \quad\longrightarrow\quad \xi\{\Gamma, y : A \vdash t\{y/x\} : B\}$$

and in the more complex case where the variable being renamed is applied to an argument through the `let` construct:

$$\mathtt{sub}\,\dfrac{\mathtt{let}\,\dfrac{\mathtt{var}\,\dfrac{\xi\{\Gamma', \Delta', w : \{\,\kappa\,\} \triangleright E \vdash v : B\}}{\xi\{\Gamma', \Delta', w : \{\, y : C \to D \vdash y : C \to D \parallel \kappa\,\} \triangleright E \vdash v : B\}} \qquad \dfrac{\xi\{\Gamma', \Delta', z : \{\, y : C \to D \vdash y : C \to D \parallel \Delta', \Psi' \vdash u : C\,\} \triangleright D \vdash t : B\}}{\xi\{\Gamma, \Delta, \Psi, x : \{\, y : C \to D \vdash y : C \to D\,\} \triangleright C \to D \vdash \mathtt{let}\ z = x\ u\ \mathtt{in}\ t : B\}}\,\mathscr{D}\parallel}{\xi\{(\Gamma, \Delta, \Psi) \ni y : C \to D \vdash (\mathtt{let}\ z = x\ u\ \mathtt{in}\ t)[x \leftarrow y] : B\}}$$

where the assumption on x appears either in Γ', in Δ' or in Ψ' — which are other identical to Γ, Δ and Ψ —, or in none of them, and the derivation is turned into:

$$\mathtt{let}\,\dfrac{\xi\{\Gamma, \Delta, w : \{\,\kappa'\,\} \triangleright E \vdash v\{y/x\} : B\} \qquad \dfrac{\xi\{\Gamma, \Delta, z : \{\, \Delta, \Psi \vdash u\{y/x\} : C\,\} \triangleright D \vdash t\{y/x\} : B\}}{\xi\{(\Gamma, \Delta, \Psi) \ni y : C \to D \vdash \mathtt{let}\ z = y\ u\{y/x\}\ \mathtt{in}\ t\{y/x\} : B\}}\,\mathscr{D}'\parallel}{\xi\{(\Gamma, \Delta, \Psi) \ni y : C \to D \vdash \mathtt{let}\ z = y\ u\{y/x\}\ \mathtt{in}\ t\{y/x\} : B\}}$$

2. The reduction rule $x[x \leftarrow u] \longrightarrow_{\mathtt{var}} u$ corresponds to the other interaction between a cut and an identity instance, and is seen on typing derivations in the following rewriting:

$$\mathtt{sub}\,\dfrac{\mathtt{var}\,\dfrac{\xi\{\,\}}{\xi\{x : A \vdash x : A\}}\,\mathscr{D}\parallel}{\xi\{x : \{\, \Psi \vdash u : A\,\} \triangleright A \vdash x : A\}}}{\xi\{\Psi \vdash x[x \leftarrow u] : A\}} \quad\longrightarrow\quad \dfrac{\xi\{\,\}}{\xi\{\Psi \vdash u : A\}}\,\mathscr{D}'\parallel$$

3. The reduction rule $t[x \leftarrow u] \longrightarrow_{\mathtt{not}} t$ corresponds to the erasure of the whole cut when it encounters a weakening, as follows:

$$\mathtt{sub}\,\dfrac{\mathtt{rem}\,\dfrac{\xi\{\Gamma, \Delta \vdash t : B\}}{\xi\{\Gamma, \Delta, x : \{\, \Sigma \vdash v : C\,\} \triangleright A \vdash t : B\}}\,\mathscr{D}\parallel}{\xi\{\Gamma, \Delta, \Psi \vdash t[x \leftarrow u] : B\}} \quad\longrightarrow\quad \mathtt{rem}^*\,\dfrac{\xi\{\Gamma, \Delta \vdash t : B\}}{\xi\{\Gamma, \Delta, \Psi \vdash t : B\}}$$

4. The reduction rule $(\lambda y.t)[x \leftarrow u] \longrightarrow_{\mathtt{lam}} \lambda y.t[x \leftarrow u]$ corresponds to the permutation of the cut over a right implication rule, and this case is just the straightforward adaptation of the corresponding case in $\lambda\overline{\mathtt{x}}$.

5. The rule $(\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u] \longrightarrow_{\mathtt{inl}} \mathtt{let}\ y = z\ v[x \leftarrow u]\ \mathtt{in}\ t$ and the similar rules `inr` and `inb` correspond to the permutation of a cut over an application when it does not involve the main formula of the cut, so that from the derivation:

$$\mathtt{sub}\ \dfrac{\mathtt{let}\ \dfrac{\xi\{\Gamma, \Omega, y : \{\kappa \parallel \Sigma, \Delta, x : \{\Omega, \Delta, \Psi \vdash u : A\} \triangleright A \vdash v : B\} \triangleright C \vdash t : D\}}{\xi\{\Gamma, \Sigma, \Omega, \Delta, x : \{\Omega, \Delta, \Psi \vdash u : A\} \triangleright A \vdash \mathtt{let}\ y = z\ v\ \mathtt{in}\ t : D\}}}{\xi\{(\Gamma, \Sigma, \Omega, \Delta, \Psi) \ni z : \{\kappa\} \triangleright B \to C \vdash (\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u] : D\}}$$

we obtain the following derivation:

$$\mathtt{let}\ \dfrac{\mathtt{sub}\ \dfrac{\xi\{\Gamma, \Omega, y : \{\kappa \parallel \Sigma, \Delta, x : \{\Omega, \Delta, \Psi \vdash u : A\} \triangleright A \vdash v : B\} \triangleright C \vdash t : D\}}{\xi\{\Gamma, \Omega, y : \{\kappa \parallel \Sigma, \Omega, \Delta, \Psi \vdash v[x \leftarrow u] : B\} \triangleright C \vdash t : D\}}}{\xi\{(\Gamma, \Sigma, \Omega, \Delta, \Psi) \ni z : \{\kappa\} \triangleright B \to C \vdash \mathtt{let}\ y = z\ v[x \leftarrow u]\ \mathtt{in}\ t : D\}}$$

and the cases of `inr` and `inb` correspond to the cases where the assumption on the type of x appears the other environment, or in both, respectively.

6. The rule $(\mathtt{let}\ y = x\ v\ \mathtt{in}\ t)[x \leftarrow u] \longrightarrow_{\mathtt{ins}} (\mathtt{let}\ y = z\ v\ \mathtt{in}\ t)[x \leftarrow u][z \leftarrow u]$ corresponds to the duplication of a cut that can be performed when this cut introduces a formula involved in an implication left rule immediately above but also in the proof above this instance.

7. The rule $(\mathtt{let}\ y = x\ v\ \mathtt{in}\ t)[x \leftarrow \lambda z.u] \longrightarrow_{\mathtt{apd}} t[y \leftarrow u[z \leftarrow v]][x \leftarrow \lambda z.u]$ and the rule $t[x \leftarrow \mathtt{let}\ y = z\ v\ \mathtt{in}\ u] \longrightarrow_{\mathtt{out}} \mathtt{let}\ y = z\ v\ \mathtt{in}\ t[x \leftarrow u]$ are the reflections of the interaction between a cut and a left implication rule, in two possible situations, and both cases are just direct the adaptations of the corresponding case in $\lambda\overline{\mathtt{x}}$.

The correspondence for the $\lambda\overline{\mathtt{s}}$-calculus is established by the same analysis of each reduction case, and the precise transformations are very similar to the ones used for $\lambda\overline{\mathtt{e}}$, simply omitting the part of the context being duplicated in the typing rules for substitutions and applications. The correspondence between cases of cut elimination and reduction rules leads to the expect theorem, stating that well-typed terms can be normalised, by cut elimination in $\mathsf{JNb} \cup \{\mathsf{ebs}\}$.

Theorem 2.3. *Given a $\lambda\overline{\mathtt{e}}$-term t well-typed in $\mathtt{N}\overline{\mathtt{e}}$, there exists a terminating strategy that allows to compute the strong normal form of t for $\longrightarrow_{\lambda\overline{\mathtt{e}}}$.*

Proof. By detour elimination in $\mathsf{JNb} \cup \{\mathsf{ebs}\}$, since it was proved that all cuts can be eliminated from a given proof in this system. Through the correspondence shown above, all redexes can be eliminated from the term t. □

The same result holds in $\lambda\overline{\mathtt{s}}$, through the correspondence of typing derivations in $\mathtt{N}\overline{\mathtt{s}}$ and proofs in the $\mathsf{JNs} \cup \{\mathsf{es}\}$ system, and the observation that elimination steps yield the reduction rules of the $\longrightarrow_{\lambda\overline{\mathtt{s}}}$ system.

3 Typing with the Calculus of Structures

The nested type systems we have considered so far were based on sequents, with typing rules that affect both the left-hand side and the right-hand side of a typing judgement, in a left/right symmetry characteristic of systems in sequent style. As a consequence, they were restricted to a particular kind of λ-calculi, the ones using pure explicit substitutions. However, standard explicit substitutions calculi, where β-redexes coexist with explicit substitutions, are much better understood. If nested type systems are meant to offer new possibilities, on the computational side, then we need to be able to handle such standard calculi.

We have seen in Chapter 2 that pure explicit substitutions λ-calculi are handled with systems based on a sequent style, while standard calculi correspond to systems of natural deduction, with introduction and elimination rules. In this situation, the calculus of structures can be used as a basis for both kind of type systems, since we can use either sequent style or natural deduction style, as shown in Chapter 4.

The problem with the $\mathsf{JS} \cup \{\mathsf{u}\}$ system is that it has no distinction between the logical implication connective and the meta-logical implication, which is typical of the calculus of structures but is problematic in the traditional typing setting — this prevents to have a typing rule for abstraction, since curryfication is implicit. Having already a nested type system for pure explicit substitutions, we are more interested in a system in natural deduction style, such as JD, to type standard λ-calculi. As in the case of sequent style systems, different variants of JD correspond to different calculi, from the most basic to more refined calculi.

3.1 Uniform Typing Structures

In a type system based on a variant of JD, judgements correspond to structures and we have to use a kind of typing judgement where different levels are nested. This is achieved by a generalisation of nested typing judgements where judgements can appear not only inside other judgements, but inside types as well. We call *uniform typing structures* such constructions, because they treat types and typing hypotheses in uniform way, which requires a definition slightly different from the definition of nested typing sequents.

Definition 3.1. *A* uniform typing structure $\mathscr{U}$*, associating a conditional type* T *to a term* t*, under a certain typing environment, is generated by the following grammar:*

$$\mathscr{U} ::= x : \phi \vdash \mathscr{U} \mid t : T \mid \emptyset \qquad T ::= A \mid \phi \to T \qquad \phi ::= \{\mathscr{U}\} \triangleright A$$

In this setting, all the notions involved are defined syntactically, in a mutually inductive way, so that there is no clear distinctions left: judgements must depend on types, but types can themselves depend on judgements. Although it might seem related, this is completely different from *dependent types*, since the distinction is kept between the terms of the object language, which is a λ-calculus, and the level of typing. Notice that we reuse the symbols from the previous section, to keep the typing rules readable — that means, as close as possible to the standard syntax for typing. We will denote typing structures by letters such as $\mathscr{U}$.

Remark 3.2. *For the sake of simplicity, we consider uniform typing structures through a set of equations, corresponding to the congruence on intuitionistic structures in the* JD *system. These equations are:*

$$x : \phi \vdash (y : \psi \vdash \mathscr{U}) \;\equiv\; y : \psi \vdash (x : \phi \vdash \mathscr{U}) \;\equiv\; x : \phi, y : \psi \vdash \mathscr{U}$$

$$\begin{array}{rcll} x : \{\, \mathscr{U}_1 \,\} \triangleright A \vdash \mathscr{U}_0 & \equiv & x : \{\, \mathscr{U}_2 \,\} \triangleright A \vdash \mathscr{U}_0 & \text{if } \mathscr{U}_1 \equiv \mathscr{U}_2 \\ t : (\{\, \mathscr{U}_1 \,\} \triangleright A) \to T & \equiv & t : (\{\, \mathscr{U}_2 \,\} \triangleright A) \to T & \text{if } \mathscr{U}_1 \equiv \mathscr{U}_2 \\ t : \phi \to T_1 & \equiv & t : \phi \to T_2 & \text{if } t : T_1 \equiv t : T_2 \end{array}$$

and they allow to use such a typing structure the same way as we could use a nested typing sequent, as if the » environment « *was a multiset of typing assumptions. Also note that we can write* $x_1 : \phi_1, \cdots, x_n : \phi_n \vdash t : A$ *in general — where* A *might be a conditional type — and we can also write* $\vdash t : A$*, if there is no typing assumption, and* $x : A$ *for an assumption that is really* $x : \{\, \emptyset \,\} \triangleright A$*.*

The main novelty lies in this notion of *conditional type*, which allows to express the fact that a particular part of a type is only valid under the condition that another typing judgement is valid. Because of the shape of the inference rules in JD and in natural deduction style systems in general, such a condition can appear before that part of the type is used, as an assumption — therefore, it must be allowed to appear within types, on the left of an implication. Notice that a conditional type is always located in negative position, and never directly on the right of a plain type — that is, a type where no condition appears.

Finally, the notion of context is slightly more complicated than in the case of nested typing sequents, because there are two different cases to handle.

Definition 3.3. *A uniform* context *is a uniform typing structure with a hole* $\{\ \}$ *meant to be filled by another uniform typing structure, as defined by the following grammar:*

$$\xi \;::=\; \{\ \} \;\mid\; x : \{\, \xi \,\} \triangleright A \vdash \mathscr{U} \;\mid\; t : (\{\, \xi \,\} \triangleright A) \to T$$

The additional case corresponds to the possibility to apply some inference rule in a conditional type, before this part of the type is moved to the left to be used as an assumption.

3.2 Uniform Typing for λ-calculi with Explicit substitutions

The purpose of uniform typing structures is to provide a framework for the nested typing of standard λ-terms with explicit substitutions, where β-redexes coexist with explicit substitutions, as in the $\lambda\mathtt{x}$-calculus and its variants.

The uniform type system used for $\lambda\mathtt{x}$ is called $\mathtt{Nx}$, and its typing rules are given in Figure 4. As before, rules can be applied inside any valid context. This system is very similar to the $\mathtt{N}\overline{\mathtt{x}}$ system based on sequents, with the significant difference that the shape of the application rule makes it correspond to a standard application $t\ u$ of one term to another. In this rule, the argument u to the function t is moved inside a conditional type, so that its type can be used as the input type of t, while retaining the condition that this is valid only if u is well-typed. As in the $\mathtt{N}\overline{\mathtt{x}}$ system, there is one rule for each construct of the language, and typing is syntax-directed.

$$\texttt{var}\ \frac{}{\Gamma, x:A \vdash x:A}\emptyset \qquad \texttt{sub}\ \frac{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash t:B}{\Gamma \vdash t[x \leftarrow u]:B}$$

$$\texttt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \texttt{app}\ \frac{\Gamma \vdash t:(\{\Gamma \vdash u:A\} \triangleright A) \to B}{\Gamma \vdash t\ u:B}$$

Figure 4: Nested type system Nx for the λx-calculus

This type system is the closest equivalent we can derive in the nested setting of the most standard S type system for the λ-calculus presented in Chapter 2 — more precisely, this would be the equivalent of the Sx system, since it is based on explicit substitutions. The sub and app rules exhibit two particular constructs of a uniform type system: a judgement nested inside a typing assumption, and the attachment of another judgement to a part of the type assigned to a term.

Remark 3.4. *The notations used in the figure above are meant to emphasise strong similarities between the* Nx *system and* Sx*, in particular by showing there the whole set* Γ *of typing assumptions, and by leaving implicit the context in which each rule can be plugged. Notice that there is always a unique judgement inside some condition, whereas there were potentially several nested typing sequents in all the systems of the previous sections.*

Example 3.5. *Below is shown an example of a typing derivation in the* Nx *system, for a* λx*-term considered under an empty set of assumptions. The simple treatment of erasure and duplication in this system leaves several assumptions in the topmost judgement — and some typing hypotheses are used several times, at several levels — but all the unnecessary assumptions are erased by the* var *rule.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z:B, y:\{\emptyset\} \triangleright B, x:\{\cdots \vdash u:A\} \triangleright A \vdash y:B}\texttt{var}}{z:B, y:\{\emptyset\} \triangleright B \vdash \lambda x.y:(\{\cdots \vdash u:A\} \triangleright A) \to B}\texttt{lam}}{z:B, y:\{z:B \vdash z:B\} \triangleright B \vdash \lambda x.y:(\{\cdots \vdash u:A\} \triangleright A) \to B}\texttt{var}}{z:B, y:\{z:B \vdash z:B\} \triangleright B \vdash (\lambda x.y)\,u:B}\texttt{app}}{z:B \vdash ((\lambda x.y)\,u)[y \leftarrow z]:B}\texttt{sub}}{\vdash \lambda z.((\lambda x.y)\,u)[y \leftarrow z]:B \to B}\texttt{lam}$$

Notice that what would be located in different branches in a more standard setting is separated here by the nesting of judgements inside conditions. Moreover, the erasure of the judgement for the subterm u, performed by the topmost var *rule, could not happen below the use of the* lam *rule, since the structure of types cannot be manipulated by anything else than the* lam *and* app *rules — and there can be no conditional type in the set of assumptions, on the left.*

$$\texttt{var}\ \frac{}{x:A \vdash x:A} \qquad \texttt{sub}\ \frac{\Gamma, \Delta, x:\{\,\Delta, \Psi \vdash u:A\,\} \triangleright A \vdash t:B}{\Gamma, \Delta, \Psi \vdash t[x \leftarrow u]:B}$$

$$\texttt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t:A \to B} \qquad \texttt{app}\ \frac{\Gamma, \Delta \vdash t:(\{\,\Delta, \Psi \vdash u:A\,\} \triangleright A) \to B}{\Gamma, \Delta, \Psi \vdash t\ u:B}$$

$$\texttt{rem}\ \frac{\Gamma \vdash t:B}{\Gamma, x:\{\,\mathscr{U}\,\} \triangleright A \vdash t:B}$$

Figure 5: Nested type system Nes for the λes-calculus

Following the refinement of λ-calculi with explicit substitutions, we can refine the Nx system to avoid the duplication of all assumptions each time an application is encountered. This leads to the Nes type system for the λes-calculus, where the erasure of assumptions is handled separatly by the rem rule and the distribution of assumptions in the sub and app rules is hybrid, only a part of the context being duplicated. The typing rules for Nes are presented in Figure 5, and one can notice that the rule for application is much simpler than the one used in the pure explicit substitution variant Nē, regarding the contents of the multisets used to hold typing assumptions.

Example 3.6. *Below is shown an example of a typing derivation in the* Nes *system, where a given typing assumption, with a condition, is unnecessary, and no assumption is ever duplicated. Note that when the assumption on a is erased, the judgement nested inside is also erased.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{w:(A\to B)\to C \vdash w:(\{\emptyset\}\triangleright A\to B)\to C}\,\texttt{var}}{w:(A\to B)\to C \vdash w:(\{\,y:A\to B \vdash y:A\to B\,\}\triangleright A\to B)\to C}\,\texttt{var}}{w:(A\to B)\to C, y:\{\emptyset\}\triangleright A\to B \vdash w\ y:C}\,\texttt{app}}{w:(A\to B)\to C, y:\{\,x:B \vdash x:B\,\}\triangleright A\to B \vdash w\ y:C}\,\texttt{var}}{w:(A\to B)\to C, y:\{\,x:B, z:A \vdash x:B\,\}\triangleright A\to B \vdash w\ y:C}\,\texttt{rem}}{w:(A\to B)\to C, y:\{\,x:B \vdash \lambda z.x:A\to B\,\}\triangleright A\to B \vdash w\ y:C}\,\texttt{lam}}{w:(A\to B)\to C, x:B \vdash (w\ y)[y\leftarrow\lambda z.x]:C}\,\texttt{sub}}{w:(A\to B)\to C \vdash \lambda x.(w\ y)[y\leftarrow\lambda z.x]:B\to C}\,\texttt{lam}}{w:(A\to B)\to C, a:\{\,\Delta \vdash u:D\,\}\triangleright E \vdash \lambda x.(w\ y)[y\leftarrow\lambda z.x]:B\to C}\,\texttt{rem}$$

When the assumption on z is erased inside the assumption on y, no inner judgement needs to be duplicated, since z was attributed the plain type A. This erasure needs to be performed before the application of the var *rule, since this axiom rule has been modified to handle exactly one assumption.*

$$\texttt{var}\ \frac{}{x:A \vdash x:A} \qquad \texttt{sub}\ \frac{\Gamma, x:\{\Delta \vdash u:A\} \triangleright A \vdash t:B}{\Gamma, \Delta \vdash t[x \leftarrow u]:B}$$

$$\texttt{lam}\ \frac{\Gamma, x:A \vdash t:B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \texttt{app}\ \frac{\Gamma \vdash t : (\{\Delta \vdash u:A\} \triangleright A) \to B}{\Gamma, \Delta \vdash t\ u : B}$$

$$\texttt{rem}\ \frac{\Gamma \vdash t:B}{\Gamma, x:\{\mathscr{U}\} \triangleright A \vdash t:B} \qquad \texttt{dup}\ \frac{\Gamma, x:\{\mathscr{U}\} \triangleright A, y:\{\mathscr{U}\} \triangleright A \vdash t_{[y/x]}:B}{\Gamma, x:\{\mathscr{U}\} \triangleright A \vdash t:B}$$

Figure 6: Nested type system $\texttt{Ns}$ for the $\lambda\texttt{s}$-calculus

Finally, we can push further the decomposition of typing rules to reach a type system suitable for the $\lambda\texttt{s}$-calculus, where duplication is performed by a separate rule — which renames an occurrence of some variable when duplicating the explicit substitution binding it. The typing rule for this system, called $\texttt{Ns}$, are shown above in Figure 6.

Remark 3.7. *Similarly to the situation in the $\texttt{N}\overline{\texttt{s}}$ system, the typing rule* dup *must be completed here with the condition that the variable x should appear at least twice in the term t, as this could otherwise lead to an infinite loop during typing.*

Example 3.8. *Below is shown an example of a derivation in the refined type system $\texttt{Ns}$, where the assumption on x, relying on an assumption on y, has to be duplicated. The duplication is performed within the typing structure and involves the renaming of an occurrence of x into w in the term being typed.*

$$\texttt{sub}\ \frac{\texttt{dup}\ \dfrac{\texttt{var}\ \dfrac{\texttt{app}\ \dfrac{\texttt{var}\ \dfrac{\texttt{app}\ \dfrac{\texttt{var}\ \dfrac{\texttt{var}\ \dfrac{\texttt{var}\ \dfrac{\emptyset}{z:A\to A\to B \vdash z:A\to(\{\emptyset\}\triangleright A)\to B}}{z:A\to A\to B \vdash z:(\{\emptyset\}\triangleright A)\to(\{x:A\vdash x:A\}\triangleright A)\to B}}{z:A\to A\to B \vdash z:(\{w:A\vdash w:A\}\triangleright A)\to(\{x:A\vdash x:A\}\triangleright A)\to B}}{z:A\to A\to B, w:A \vdash z\ w:(\{x:\{\emptyset\}\triangleright A\vdash x:A\}\triangleright A)\to B}}{z:A\to A\to B, w:A \vdash z\ w:(\{x:\{y:A\vdash y:A\}\triangleright A\vdash x:A\}\triangleright A)\to B}}{z:A\to A\to B, w:\{\emptyset\}\triangleright A, x:\{y:A\vdash y:A\}\triangleright A\vdash z\ w\ x:B}}{z:A\to A\to B, w:\{y:A\vdash y:A\}\triangleright A, x:\{y:A\vdash y:A\}\triangleright A\vdash z\ w\ x:B}}{z:A\to A\to B, x:\{y:A\vdash y:A\}\triangleright A\vdash z\ x\ x:B}}{y:A, z:A\to A\to B\vdash(z\ x\ x)[x\leftarrow y]:A\to B}$$

Notice that the var *rule has to be applied twice on $y:A$ because all the contents of the condition in the assumption on x is also duplicated along the way. This could be avoided by using the* var *rule there before the* dup *rule causing the duplication but it illustrates how parts of the typing process can be performed redundantly in this setting.*

3.3 Properties of Nested Typing with Structures

The nested type systems presented for standard λ-calculi with explicit substitutions use a slightly more complex syntax than the ones based on nested sequents, so that the good properties proved in the previous section must be adapted to fit the setting of uniform typing structures. In particular, the way uniform typing structures are defined is more modular than the definition of nested typing sequents, but it turns out to be difficult to accomodate in the proof of termination the equations defined used on structures. The part of the measure concerning only terms is easily defined following the same scheme as before:

Definition 3.9. *The* weight *of a λx-term t is defined recursively as:*

$$\begin{array}{rlrl} \mathtt{W}(x) &= 1 & \mathtt{W}(u\,v) &= 1+\mathtt{W}(u)+\mathtt{W}(u)\times\mathtt{W}(v) \\ \mathtt{W}(\lambda x.u) &= 1+\mathtt{W}(u) & \mathtt{W}(u[x\leftarrow v]) &= 1+\mathtt{W}(u)+\mathtt{W}(u)\times\mathtt{W}(v) \end{array}$$

Notice that the measure required for the standard form of application is simpler than the one we used before, as it was induced by the duplication of the assumption on the variable applied. In order to extend the notion of weight to uniform typing structure, we will adopt the radical solution of disregarding the equations between them, by using a notation closer to the nested sequents syntax. Just as in the notation used in the typing rules of $\mathtt{Nx}$, we consider maximal structures and write them in the form of:

$$x_1:\phi_1,\cdots,x_n:\phi_n \vdash t:A$$

where there might be no ϕ_i, in which case the antecedent is denoted $\emptyset$, and A is a conditional type that can contain typing structures. The presence of conditions within types is another reason for the technicality of the proof of termination. Then, the definition of the weight can be extended to all other notions involved in the type system, and the notation for it is overloaded.

Definition 3.10. *Given a uniform typing structure $\mathscr{U}$ for the λx-calculus, the* weight *of $\mathscr{U}$ is denoted by $\mathtt{W}(\mathscr{U})$ and defined by induction as:*

$$\mathtt{W}(\emptyset) = 0 \quad \textit{and} \quad \mathtt{W}(\Gamma\vdash t:A) = (\mathtt{W}(\Gamma)+\mathtt{W}(A))\times\mathtt{W}(t)$$

where the weight $\mathtt{W}(\Gamma)$ of a sequence of typing assumptions Γ is defined as:

$$\mathtt{W}(\emptyset) = 1 \quad \textit{and} \quad \mathtt{W}(x:\{\,\mathscr{U}_x\,\}\triangleright B,\Delta) = \mathtt{W}(\mathscr{U}_x)+\mathtt{W}(\Delta)$$

and the weight $\mathtt{W}(A)$ of a conditional type A is defined as:

$$\mathtt{W}(\{\,\mathscr{U}_A\,\}\triangleright B\to C) = \mathtt{W}(\mathscr{U}_A)+\mathtt{W}(C) \quad \textit{and} \quad \mathtt{W}(A) = 0 \quad \textit{if A is a plain type}$$

Through all these definitions, the notion of weight used for $\mathtt{Nx}$ is similar to the one used in $\mathtt{N\overline{x}}$, but the presence of conditions inside types forces to take the type assigned to a term into account. We can now prove termination by inspecting the typing rules of $\mathtt{Nx}$, which are all such that the weight of the involved uniform typing structure decreases from conclusion to premise.

Theorem 3.11. *The typing process in* $\mathtt{Nx}$ *is terminating.*

Proof. For a $\lambda\mathtt{x}$-term t under a typing environment Γ, to which we want to assign type A, we proceed by induction on $\mathtt{W}(\Gamma \vdash t : A)$. In the base case, only the `var` rule can be applied on t, which is a variable, and typing terminates. In the general case, we consider any term u under an environment Δ to which we can apply a typing rule — if there is none, then typing fails and thus terminates immediately. Notice that this u can be either t, or any other term to be typed inside a condition in Γ.

Once this term u is chosen, we use a case analysis on the typing rules that can be applied to it, and in most cases they are of the shape:

$$\mathsf{r}\,\frac{\Phi \vdash v : C}{\Delta \vdash u : B} \qquad \equiv \qquad \mathsf{r}\,\frac{\mathscr{U}_C}{\mathscr{U}_B}$$

where v is either t or a subterm of t, and Φ is a modification of Γ. In each case, we show that we have $\mathtt{W}(\mathscr{U}_C) < \mathtt{W}(\mathscr{U}_B)$, as follows:

1. If u is a variable x, we apply the `var` rule which has no premise, so that this term x appearing in a condition is removed from Δ and thus $\mathtt{W}(\mathscr{U}_C) < \mathtt{W}(\mathscr{U}_B)$.

2. If u of the shape $\lambda x.p$ and B is $D \to E$, then if D is a plain type, the result is easy, and otherwise we have $\mathtt{W}(\Phi) = \mathtt{W}(\Delta) + k$, where k is the weight of the structure in D, so that $\mathtt{W}(\mathscr{U}_C) = (\mathtt{W}(\Delta) + k + \mathtt{W}(E)) \times \mathtt{W}(p)$ while the conclusion is such that $\mathtt{W}(\mathscr{U}_B) = (\mathtt{W}(\Delta) + \mathtt{W}(D \to E)) \times (1 + \mathtt{W}(p))$ — and we conclude by the observation that $\mathtt{W}(D \to E) = k + \mathtt{W}(E)$, by definition.

3. If u is of the shape $p\ q$ then $\mathtt{W}(\mathscr{U}_B) = (\mathtt{W}(\Delta) + \mathtt{W}(B)) \times (1 + \mathtt{W}(p) + \mathtt{W}(p) \times \mathtt{W}(q))$ and $\mathtt{W}(\mathscr{U}_C) = (\mathtt{W}(\Delta) + \mathtt{W}(\Delta) \times \mathtt{W}(q) + \mathtt{W}(B)) \times \mathtt{W}(p)$, since the type for q that was introduced by `app` is plain and has weight 0, so that $\mathtt{W}(\mathscr{U}_C) < \mathtt{W}(\mathscr{U}_B)$.

4. If u is of the shape $p[x \leftarrow q]$, then the weight $\mathtt{W}(\mathscr{U}_B)$ has the same value as in the application case, and the `sub` rule produces also a typing structure $\mathscr{U}_C$ of the same weight as in the previous case, so that $\mathtt{W}(\mathscr{U}_C) < \mathtt{W}(\mathscr{U}_B)$. □

Remark 3.12. *In a practical use of such a nested type system with conditional types, the type assigned to a term would be unknown until its typing process is complete, but the structure of the type would be discovered incrementally. Types using metavariables could be used to allow the representation of an unknown types containing conditions.*

As usual, it can be shown that to a given $\lambda\mathtt{x}$-term, the $\mathtt{Nx}$ system can only assign one unique type, up to renaming. The type assigned is always a plain type, because the conditions are only introduced in the premise of the `app` rule.

Proposition 3.13. *For any typing environment* Γ *and any* $\lambda\overline{\mathtt{x}}$*-term* t*, there is at most one type* A *such that* $\Gamma \vdash t : A$ *in* $\mathtt{N}\overline{\mathtt{x}}$*, up to renaming of base types in* A*.*

Finally, the $\mathtt{Nes}$ system can be treated the same way, and the adaptation of the technique to the $\mathtt{Ns}$ system requires the same adjustments of measure than the $\mathtt{N}\overline{\mathtt{s}}$ system, using multiset ordering and the number of occurrences of variables in the terms being typed.

4 Detour Elimination as Reduction

Following the same methodology as in the case of nested sequents, we show how the detour elimination process in the variants of the JD system can be interpreted as reduction in standard λ-calculi with explicit substitutions, by the correspondence between proofs and typing derivations. This procedure is simpler than the one for cut elimination in nested sequents, because it never requires a non-local rewriting of the proof, so that the modifications on typing derivations during reduction are always local to the part of the term being reduced.

We start by establishing the correspondence between detour elimination in the JDa system and reduction in the basic λx-calculus. Then, we will explain how this correspondence can be extended to more refined proof systems and calculi.

4.1 Reduction in the λx-calculus

As with λ-calculi with explicit substitutions, or standard shallow type systems, we describe the correspondence between the dynamics of λx and detour elimination in JDa by considering each reduction rule of $\longrightarrow_{\lambda\mathtt{x}}$:

1. The reduction rule $(\lambda x.t)\,u \longrightarrow_{\mathtt{B}} t[x \leftarrow u]$ corresponds to the transformation of a matching pair of introduction and elimination rules into a cut instance:

$$\begin{array}{l}\mathtt{lam}\\ \mathtt{app}\end{array}\dfrac{\dfrac{\xi\{\Gamma, x:\{\Gamma\vdash u:A\}\triangleright A\vdash t:B\}}{\xi\{\Gamma\vdash \lambda x.t:(\{\Gamma\vdash u:A\}\triangleright A)\to B\}}}{\xi\{\Gamma\vdash(\lambda x.t)\,u:B\}} \quad\longrightarrow\quad \mathtt{sub}\dfrac{\xi\{\Gamma, x:\{\Gamma\vdash u:A\}\triangleright A\vdash t:B\}}{\xi\{\Gamma\vdash t[x\leftarrow u]:B\}}$$

2. The reduction rule $x[x \leftarrow u] \longrightarrow_{\mathtt{var}} u$ corresponds to the interaction of a cut with an axiom instance, and is seen on typing derivations as follows:

$$\mathtt{sub}\dfrac{\mathtt{var}\dfrac{\xi\{\emptyset\}}{\xi\{\Gamma, x:\{\emptyset\}\triangleright A\vdash x:A\}}\quad \begin{array}{c}\mathscr{D}_1\|\\ \xi\{\Gamma, x:\{\Gamma\vdash u:A\}\triangleright A\vdash x:A\}\end{array}}{\xi\{\Gamma\vdash x[x\leftarrow u]:A\}} \quad\longrightarrow\quad \begin{array}{c}\xi\{\emptyset\}\\ \mathscr{D}_1'\|\\ \xi\{\Gamma\vdash u:A\}\end{array}$$

   where $\mathscr{D}_1'$ is obtained by extracting $\mathscr{D}_1$ from its toplevel context.

3. The reduction rule $z[x \leftarrow u] \longrightarrow_{\mathtt{nov}} z$ corresponds to the erasure of the whole cut when it encounters an implicit weakening, as shown below:

$$\mathtt{sub}\dfrac{\mathtt{var}\dfrac{\xi\{\emptyset\}}{\xi\{\Gamma, x:\{\mathscr{U}\}\triangleright A\vdash z:B\}}\quad \begin{array}{c}\mathscr{D}_1\|\\ \xi\{\Gamma, x:\{\Gamma\vdash u:A\}\triangleright A\vdash z:B\}\end{array}}{\xi\{\Gamma\ni z:B\vdash z[x\leftarrow u]:B\}} \quad\longrightarrow\quad \mathtt{var}\dfrac{\xi\{\emptyset\}}{\xi\{\Gamma\ni z:B\vdash z:B\}}$$

4. The reduction rule $(\lambda y.t)[x \leftarrow u] \longrightarrow_{\mathtt{lam}} \lambda y.t[x \leftarrow u]$ corresponds to the permutation of a cut over an introduction, so that from the derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{lam}\,\dfrac{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A, y:B \vdash t:C\}}{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash \lambda y.t:B \to C\}}}{\xi\{\Gamma \vdash (\lambda y.t)[x \leftarrow u]:B \to C\}}$$

we obtain the following new derivation:

$$\mathtt{lam}\,\dfrac{\mathtt{sub}\,\dfrac{\xi\{\Gamma, x:\{\Gamma, y:B \vdash u:A\} \triangleright A, y:B \vdash t:C\}}{\xi\{\Gamma, y:B \vdash t[x \leftarrow u]:C\}}}{\xi\{\Gamma \vdash \lambda y.t[x \leftarrow u]:B \to C\}}$$

5. The reduction rule $(t\ v)[x \leftarrow u] \longrightarrow_{\mathtt{app}} t[x \leftarrow u]\ v[x \leftarrow u]$ corresponds to the permutation of a cut over an elimination rule, so that from the following derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{app}\,\dfrac{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash t:(\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash v:B\} \triangleright B) \to C\}}{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash t\ v:C\}}}{\xi\{\Gamma \vdash (t\ v)[x \leftarrow u]:C\}}$$

we obtain the following new derivation:

$$\mathtt{sub}\,\dfrac{\mathtt{sub}\,\dfrac{\mathtt{app}\,\dfrac{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash t:(\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash v:B\} \triangleright B) \to C\}}{\xi\{\Gamma, x:\{\Gamma \vdash u:A\} \triangleright A \vdash t:(\{\Gamma \vdash v[x \leftarrow u]:B\} \triangleright B) \to C\}}}{\xi\{\Gamma \vdash t[x \leftarrow u]:(\{\Gamma \vdash v[x \leftarrow u]:B\} \triangleright B) \to C\}}}{\xi\{\Gamma \vdash t[x \leftarrow u]\ v[x \leftarrow u]:C\}}$$

4.2 Reduction in Other Calculi

As before, the correspondence established for $\lambda\mathtt{x}$ can be refined to obtain the other correspondences between variant calculi and more refined proof systems. The two sets of typing rules for the $\lambda\mathtt{es}$ and $\lambda\mathtt{s}$ calculi are quite similar, and we show here only the correspondence for reduction in the $\lambda\mathtt{s}$-calculus. The reduction cases for the $\lambda\mathtt{es}$-calculus are obtained by considering possible duplications in substitution and application typing rules.

1. The reduction rule $(\lambda x.t)\,u \longrightarrow_{\mathtt{B}} t[x \leftarrow u]$ corresponds to the transformation of a matching pair of introduction and elimination rules into a cut instance:

$$\mathtt{app}\,\dfrac{\mathtt{lam}\,\dfrac{\xi\{\Gamma, x:\{\Delta \vdash u:A\} \triangleright A \vdash t:B\}}{\xi\{\Gamma \vdash \lambda x.t:(\{\Delta \vdash u:A\} \triangleright A) \to B\}} \quad \xi\{\Gamma, \Delta \vdash (\lambda x.t)\,u:B\}} \quad\longrightarrow\quad \mathtt{sub}\,\dfrac{\xi\{\Gamma, x:\{\Delta \vdash u:A\} \triangleright A \vdash t:B\}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u]:B\}}$$

2. The reduction rule $x[x \leftarrow u] \longrightarrow_{\mathtt{var}} u$ corresponds to the interaction of a cut with an axiom, and it is almost the same as the corresponding case in $\lambda\mathtt{x}$.

3. The reduction rule $t[x \leftarrow u] \longrightarrow_{\mathtt{not}} t$ corresponds to the erasure of the whole cut when it encounters a weakening, as shown below:

$$\mathtt{rem}\,\dfrac{\xi\{\Gamma \vdash t : B\}}{\xi\{\Gamma, x : \{\mathscr{U}\} \triangleright A \vdash t : B\}} \quad \begin{array}{c} \mathscr{D}_1 \| \\ \mathtt{sub}\,\dfrac{\xi\{\Gamma, x : \{\Delta \vdash u : A\} \triangleright A \vdash t : B\}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u] : B\}} \end{array} \quad \longrightarrow \quad \mathtt{rem}^*\,\dfrac{\xi\{\Gamma \vdash t : B\}}{\xi\{\Gamma, \Delta \vdash t : B\}}$$

4. The reduction rule $t[x \leftarrow u] \longrightarrow_{\mathtt{dup}} t_{[y/x]}[x \leftarrow u][y \leftarrow u]$ corresponds to the duplication of the whole cut when it encounters a contraction, so that from the following derivation:

$$\begin{array}{c} \mathtt{dup}\,\dfrac{\xi\{\Gamma, x : \{\mathscr{U}\} \triangleright A, y : \{\mathscr{U}\} \triangleright A \vdash t : B\}}{\xi\{\Gamma, x : \{\mathscr{U}\} \triangleright A \vdash t : B\}} \\ \mathscr{D}_1 \| \\ \mathtt{sub}\,\dfrac{\xi\{\Gamma, x : \{\Delta \vdash u : A\} \triangleright A \vdash t : B\}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u] : B\}} \end{array}$$

we obtain:

$$\begin{array}{c} \xi\{\Gamma, y : \{\mathscr{U}\} \triangleright A, x : \{\mathscr{U}\} \triangleright A \vdash t_{[y/x]} : B\} \\ \mathscr{D}'_1 \| \\ \xi\{\Gamma, y : \{\mathscr{U}\} \triangleright A, x : \{\Delta \vdash u : A\} \triangleright A \vdash t_{[y/x]} : B\} \\ \mathscr{D}'_1 \| \\ \mathtt{sub}\,\dfrac{\xi\{\Gamma, y : \{\Delta \vdash u : A\} \triangleright A, x : \{\Delta \vdash u : A\} \triangleright A \vdash t_{[y/x]} : B\}}{\mathtt{sub}\,\dfrac{\xi\{\Gamma, \Delta, y : \{\Delta \vdash u : A\} \triangleright A \vdash t_{[y/x]}[x \leftarrow u] : B\}}{\mathtt{dup}^*\,\dfrac{\xi\{\Gamma, \Delta, \Delta \vdash t_{[y/x]}[x \leftarrow u][y \leftarrow u] : B\}}{\xi\{\Gamma, \Delta \vdash t_{[y/x]}[x \leftarrow u][y \leftarrow u] : B\}}}} \end{array}$$

where $\mathscr{D}'_1$ is obtained by duplicating $\mathscr{D}_1$ and changing its context.

5. The reduction rule $(\lambda y.t)[x \leftarrow u] \longrightarrow_{\mathtt{lam}} \lambda y.t[x \leftarrow u]$ corresponds to the permutation of a cut over an introduction rule instance, it is a straightforward adaptation of the corresponding case in $\lambda\mathbf{x}$.

6. The reduction rule $(t\ v)[x \leftarrow u] \longrightarrow_{\mathtt{apl}} t[x \leftarrow u]\ v$ reflects the permutation of a cut over an elimination rule, so that from the following derivation:

$$\mathtt{app}\,\dfrac{\xi\{\Gamma, x : \{\Psi \vdash u : A\} \triangleright A \vdash t : (\{\Delta \vdash v : B\} \triangleright B) \to C\}}{\mathtt{sub}\,\dfrac{\xi\{\Gamma, \Delta, x : \{\Psi \vdash u : A\} \triangleright A \vdash t\ v : C\}}{\xi\{\Gamma, \Delta, \Psi \vdash (t\ v)[x \leftarrow u] : C\}}}$$

we obtain the following new derivation:

$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, x : \{\Psi \vdash u : A\} \triangleright A \vdash t : (\{\Delta \vdash v : B\} \triangleright B) \to C\}}{\mathtt{app}\,\dfrac{\xi\{\Gamma, \Psi \vdash t[x \leftarrow u] : (\{\Delta \vdash v : B\} \triangleright B) \to C\}}{\xi\{\Gamma, \Delta, \Psi \vdash t[x \leftarrow u]\ v : C\}}}$$

and the other rule `apr` corresponds to the other case, where the assumption on the type of x is used to type the term v, so that the `sub` rule is moved upwards inside the argument context.

Chapter 6

Nested Typing and Extended λ-calculi

In this chapter, we show how features specific to proof systems in the calculus of structures can be used to type variants of the λ-calculus with explicit substitutions extended by new operators, expressing a complex control of reduction. Indeed, the type systems presented in the previous chapter were following the scheme used in standard type systems, so that they can be used only to type λ-calculi with standard operations — either in the standard style or using sequentialised application. More complex constructions can be handled by type systems relying on two of the most important features of systems in the calculus of structures: the switch rule and the way contraction is handled.

The particular treatment of contraction in proof systems like JD is induced by the collapse of the formulas with the meta-level in the deep inference setting. Not only an assumption — a formula on the left in a sequent — is duplicated, but also the *goal* of another sequent, a formula on the right. Through the typing approach, it means that when a variable is used twice, it can be used to represent two different pieces of the program, corresponding to the possibly different proofs given for the copies of the formulas being duplicated. Here, we explore this new expressivity by describing a λ-calculus with *resources*, where syntax supports the superposition of different terms, which may share a common context. Unfortunately, the operational behaviour of this calculus lacks good properties, in particular because of deadlocks, although typing ensures that there exists a way to normalise a well-typed term.

The switch rule has a more radical impact on the design of a λ-calculus, through the operators that are typed by this rule. As it is used in a proof system to distribute hypotheses to the different substructures, it yields *communication* operators in the λ-calculus that are responsible for distributing *computational resources* — that is, explicit substitutions — among different subterms considered as parallel threads, with a hierarchy between functions and their arguments. Reduction in this setting is much more complex than in standard calculi, and we only prove here that typing ensures the existence of a terminating reduction. The restrictions needed to recover properties such as confluence would be complex, and have a *global* aspect.

1 Contraction and Resources

In the correspondence between proof systems and λ-calculi, the contraction rule and its incarnation in other inference rules is responsible for the behaviour of terms with respect to duplication. In the reduction system, duplicating a part of the term corresponds to a rewriting in the proof that involves a contraction. As explained in Chapter 5 this is valid in the nested setting as it is in the standard setting. However, in a nested proof system, contraction affects more than a formula: it works on the representation of the meta-level as well, and therefore on what would be sequents and branches in a shallow formalism.

1.1 Contraction in Nested Proof Systems

Consider the two instances of the basic contraction rule, one from a system such as JN, in nested sequents, and the other in the calculus of structures, in JD:

$$\mathsf{c}\,\frac{\Gamma,[\Delta\vdash B],[\Delta\vdash B]\vdash A}{\Gamma,[\Delta\vdash B]\vdash A} \qquad \mathsf{c}\,\frac{(B\Rightarrow(c\to D))\Rightarrow(B\Rightarrow(c\to D))\Rightarrow A}{(B\Rightarrow(c\to D))\Rightarrow A}$$

On the left, the contraction is clearly duplicating more than the formula B, it copies an entire sequent along with all the assumptions in Δ — and all the sequents nested inside. In a proof in the sequent calculus, only the formula B would be duplicated, as shown below on the left, and the proof could be translated into nested sequents by plugging the translation of the proof of $\Delta=\Psi\vdash B$ below the contraction:

$$\mathsf{cut}\,\frac{\overset{\mathscr{P}_1}{\Psi\vdash B} \qquad \mathsf{cont}\,\dfrac{\overset{\mathscr{P}_2}{\Gamma,B,B\vdash A}}{\Gamma,B\vdash A}}{\Gamma,\Psi\vdash A} \quad\longrightarrow\quad \mathsf{es}\,\frac{\mathsf{c}\,\dfrac{\overset{\mathscr{P}_2\Vdash}{\Gamma,[\vdash B],[\vdash B]\vdash A}}{\Gamma,[\vdash B]\vdash A}}{\Gamma,\Psi\vdash A}$$

but there are also other possible translations. In particular, the proof $\mathscr{P}_1$ can be duplicated and the translation of each copy plugged above the contraction as shown below on the left. Otherwise, the contraction is located anywhere in the translation of the proof $\mathscr{P}_1$, which is separated into two parts $\mathscr{P}_3$ and $\mathscr{P}_4$, as shown below on the right:

$$\mathsf{c}\,\frac{\mathsf{es}\,\dfrac{\begin{array}{c}\mathscr{P}_2\Vdash\\ \Gamma,[\vdash B],[\vdash B]\vdash A\\ \mathscr{P}_1\Vdash\\ \Gamma,[\vdash B],[[\Psi\vdash B]\vdash B]\vdash A\\ \mathscr{P}_1\Vdash\end{array}}{\Gamma,[[\Psi\vdash B]\vdash B],[[\Psi\vdash B]\vdash B]\vdash A}}{\Gamma,[[\Psi\vdash B]\vdash B]\vdash A} \qquad \mathsf{es}\,\frac{\mathsf{c}\,\dfrac{\begin{array}{c}\mathscr{P}_2\Vdash\\ \Gamma,[\vdash B],[\vdash B]\vdash A\\ \mathscr{P}_4'\Vdash\\ \Gamma,[\vdash B],[\Sigma\vdash B]\vdash A\\ \mathscr{P}_4\Vdash\end{array}}{\Gamma,[\Sigma\vdash B],[\Sigma\vdash B]\vdash A}}{\Gamma,[\Sigma\vdash B]\vdash A}}{\Gamma,\Psi\vdash A}$$

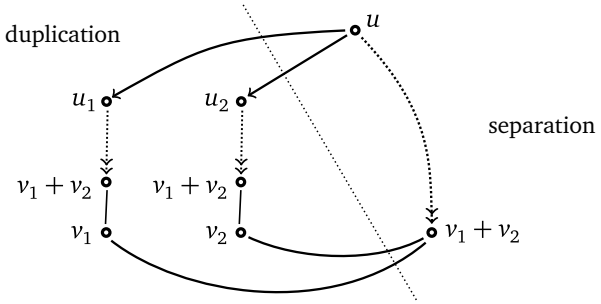
In the sequent calculus, only the two extreme options are available: either the proof $\mathscr{P}_1$ corresponds to both copies of B, or each copy of B corresponds to a copy of $\mathscr{P}_1$ — as obtained after cut elimination, when the cut is duplicated. The specificity of the nested setting, that we want to exploit, is the existence of the intermediate proofs, where a contraction separates only a part of the proof. Indeed, notice that in this situation, the two proofs $\mathscr{P}_4$ and $\mathscr{P}_4'$ can be different, so that the two copies of B have different proofs but *share* the $\mathscr{P}_3$ part.

The situation is exactly the same in the calculus of structures, and in particular in the system JD in natural deduction style. In the contraction instance from JD shown above the implication $c \to D$ corresponds to a formula, while $B \Rightarrow (c \to D)$ is the equivalent of a sequent — and duplicating this whole structure implies that B will be proved twice. We can then try to use this in a nested type system derived from a variant of JD, where interpreting contraction allows to explicitly deal with resources, and where some terms can share a common context — just as the two proofs $\mathscr{P}_4$ and $\mathscr{P}_4'$ above share the derivation $\mathscr{P}_3$.

In order to introduce a notion of *resources* in a λ-calculus where duplication of terms is allowed, we will consider two interpretations of contraction in the system, one for standard duplication and another one for the *separation* of two resources u and v packed as a term $u + v$. In order to obtain a simple calculus, we associate this operation with a variant of the contraction rule:

$$\frac{(\Gamma \Rightarrow A) \Rightarrow (\Delta \Rightarrow A) \Rightarrow B}{(\Gamma \Rightarrow \Delta \Rightarrow A) \Rightarrow B}$$

which was discussed in Chapter 4. Using this rule, the two different proofs of A use different assumptions — or copies of the same assumptions created before. In the correspondence between proofs and terms, we write a term like $C[u+v]$ of type A to interpret a proof of A splitted into $C[-]$ and the subproofs u and v. Then, the goal is to have a distinction between standard terms and terms labelled as sums, as the sharing of $C[-]$ by u and v is preserved by reduction only if a term containing a sum is not duplicated before its subterms are separated — as illustrated below.



The calculus we will present is not expressive enough to ensure that no term meant to be separated is duplicated, but it allows to define terms where the subterms to be separated and the subterms to be duplicated are distinguished enough for reduction to preserve some sharing.

1.2 Syntax of the λr-calculus

The syntax used to introduce resources in the λr-calculus is just an extension of the syntax of the λs-calculus [KR11]. It relies on the use of a *sum operator* on terms, which expresses the idea that two terms u and v in the same context $C[-]$ can be composed into one single term where the context $C[-]$ is shared among u and v, by writing $C[u+v]$. This $+$ syntax is meant to convey the idea of *superposition* of $C[u]$ and $C[v]$, and was borrowed from the *differential* λ-calculus [ER03], although it does not behave the same — indeed, the term $u+v$ does not represent different possible results of a computation, but is rather a term that can be used twice, once with the value of u and once with the value of v.

The distinction between terms considered as a superposition of two terms and the terms treated as usual is expressed within the binders, so that the treatment of an argument depends on the code of the function to which it is given. This requires to refine the notion of binding, and we will have two levels of binding names: the usual variables of the λs-calculus, that we call the *basic names*, are denoted letters such as a, b or c while the *compound names* are denoted by x, y or z.

Definition 1.1. *The language of terms of the λr-calculus is defined as:*

$$t,u ::= a \mid \lambda x.t \mid t\ u \mid t[x \leftarrow u] \mid t+u \qquad \textit{with}\ \ x,y ::= a \mid x+y$$

In comparison with the λs-calculus, the only extension is the introduction of the sum, both at the level of terms and at the level of binding names. In the terms, the sum is just another binary construct, as the application, and it can contain any two subterms. The consequence for bindings is that one abstraction, or one explicit substitution, can now bind more than one basic name at a time. The *binding set* of a compound name x, denoted by $\langle x \rangle$, is the set of all basic names x contains:

$$\langle a \rangle = \{a\} \qquad \langle x+y \rangle = \langle x \rangle \cup \langle y \rangle$$

and this implies that all these basic names should be considered when defining the sets of bound and free variables of a term. Note that any basic name a must appear at most once in a compound name x.

Definition 1.2. *Given a λr-term t, the set* $\mathrm{bv}(t)$ *of its* bound variables *is:*

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{bv}(a) & = & \emptyset & \mathrm{bv}(t\ u) & = & \mathrm{bv}(t) \cup \mathrm{bv}(u) \\ \mathrm{bv}(\lambda x.t) & = & \mathrm{bv}(t) \cup \langle x \rangle & \mathrm{bv}(t[x \leftarrow u]) & = & \mathrm{bv}(t) \cup \langle x \rangle \cup \mathrm{bv}(u) \\ \mathrm{bv}(t+u) & = & \mathrm{bv}(t) \cup \mathrm{bv}(u) & & & \end{array}$$

and the set $\mathrm{fv}(t)$ *of its* free variables *is:*

$$\begin{array}{rcl@{\qquad}rcl} & & & \mathrm{fv}(t+u) & = & \mathrm{fv}(t) \cup \mathrm{fv}(u) \\ \mathrm{fv}(a) & = & a & \mathrm{fv}(t\ u) & = & \mathrm{fv}(t) \cup \mathrm{fv}(u) \\ \mathrm{fv}(\lambda x.t) & = & \mathrm{fv}(t) \setminus \langle x \rangle & \mathrm{fv}(t[x \leftarrow u]) & = & (\mathrm{fv}(t) \setminus \langle x \rangle) \cup \mathrm{fv}(u) \end{array}$$

Remark 1.3. *The syntax of the λr-calculus defines a set of λr-terms which is a strict superset of of the set of λs-terms. Indeed, any given λs-term can be considered as a valid λr-term, if all its binding names are interpreted as basic names from λr, so that the extension of bindings and terms is » orthogonal « to the syntax of λs.*

$$
\begin{array}{rcll}
(\lambda x.t)\,u & \longrightarrow_{\mathtt{B}} & t[x \leftarrow u] & \\
a[a \leftarrow u] & \longrightarrow_{\mathtt{var}} & u & \\
t[a \leftarrow u] & \longrightarrow_{\mathtt{rem}} & t & (a \notin t) \\
t[a \leftarrow u] & \longrightarrow_{\mathtt{dup}} & t_{[b/a]}[b \leftarrow u][a \leftarrow u] & (|t|_a > 1, b \notin t) \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] & \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apl}} & t[x \leftarrow u]\ v & (x \notin v) \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apr}} & t\ v[x \leftarrow u] & (x \notin t) \\
t[y \leftarrow v][x \leftarrow u] & \longrightarrow_{\mathtt{cmp}} & t[y \leftarrow v[x \leftarrow u]] & (x \notin t, x \in v) \\
t[x + y \leftarrow u + v] & \longrightarrow_{\mathtt{sep}} & t[x \leftarrow u][y \leftarrow v] & \\
(t + v)[x \leftarrow u] & \longrightarrow_{\mathtt{sul}} & t[x \leftarrow u] + v & (x \notin v) \\
(t + v)[x \leftarrow u] & \longrightarrow_{\mathtt{sur}} & t + v[x \leftarrow u] & (x \notin t)
\end{array}
$$

$$
t[x \leftarrow u][y \leftarrow v] \equiv_e t[y \leftarrow v][x \leftarrow u] \qquad (y \notin u, x \notin v)
$$

Figure 1: Reduction rules and equation of the $\lambda\mathtt{r}$-calculus

Following usual notations, we write $a \in t$ if $a \in \mathrm{fv}(t)$ and $a \notin t$ otherwise, and this can be extended to compound names by writing $x \in t$ if and only if there exists $a \in \langle x \rangle$ such that $a \in t$, and $x \notin t$ otherwise. Then, clashes between variable names are avoided with α-conversion, using Barendregt's convention [Bar84]. Notice that in the context of compound bindings, the renaming happens only on basic names:

$$
\lambda x.t \equiv_\alpha \lambda x\{b/a\}.t\{b/a\} \qquad \text{if } b \notin x \text{ and } b \notin \mathrm{bv}(t)
$$

where $\{b/a\}$ denotes the renaming of a into b, and similarly for the binding within an explicit substitution. Moreover, we use the notation $|t|_a$ as in $\lambda\mathtt{s}$, for the number of free occurrences of a in t, and also $t_{[b/a]}$ for the non-deterministic replacement in t of one free occurrence of a by b.

The purpose of the reduction system of $\lambda\mathtt{r}$ is to perform substitution of terms for basic names, but explicit substitutions allow to handle compound names, which need to be decomposed during the process. Indeed, the *implicit substitution* $t\{u/x\}$ is only defined when x is a basic name a, and $t\{u/a\}$ is as usual the term t where all free occurrences of a have been replaced with u simultaneously.

Reduction rules. The specific reduction rules used for the $\lambda\mathtt{r}$-calculus are given above in Figure 1. Note that if we considering $\lambda\mathtt{s}$-terms, where no compound name is used in bindings and no sum appears in any subterm, the rules `sep`, `sul` and `sur` are superfluous, and the rest are rules of $\lambda\mathtt{s}$. The new rules are:

- The `sep` rule reduces a term where a substitution on $x + y$ contains a sum of the shape $u + v$, which can thus be splitted, so that x is then associated to u and y to v, in two separate explicit substitutions.
- The `sul` and `sur` rules dispatch the substitution to one side of a sum, as it would be done with an application.

The reduction system comes with the usual equation to exchange independent explicit substitutions, needed with the rule for composition to allow a substitution to be pushed independently from other substitutions. We call $\longrightarrow_{\lambda\mathtt{r}}$ the reduction defined by the rules of Figure 1, and $\longrightarrow^{\equiv}_{\lambda\mathtt{r}}$ the reduction defined by incorporating the relation $\equiv$ induced by α-conversion and $\equiv_e$, so that $t \longrightarrow^{\equiv}_{\lambda\mathtt{r}} u$ if and only if there is t' and u' such that $t \equiv t' \longrightarrow_{\lambda\mathtt{r}} u' \equiv u$.

Example 1.4. *Here is an example of a possible reduction sequence for some $\lambda\mathtt{r}$-term, where the use of the rules* `sep` *and* `sur` *is illustrated:*

$$\begin{array}{ll} & (b\ (c+a))[a+(b+c) \leftarrow (u+v)+p\ q] \\ \longrightarrow_{\mathtt{sep}} & (b\ (c+a))[a \leftarrow u+v][b+c \leftarrow p\ q] \\ \longrightarrow_{\mathtt{apr}} & (b\ (c+a)[a \leftarrow u+v])[b+c \leftarrow p\ q] \\ \longrightarrow_{\mathtt{sur}} & (b\ (c+a[a \leftarrow u+v]))[b+c \leftarrow p\ q] \\ \longrightarrow_{\mathtt{var}} & (b\ (c+(u+r)))[b+c \leftarrow p\ q] \end{array}$$

where we can see that one part of the body of the original substitution can be moved inside the sum subterm, but not the part containing the application p q, since it is not of the shape of a sum. If it is eventually reduced into a sum, then it can also be pushed inside both the sum and application subterms.

The $\lambda\mathtt{r}$-calculus can play on linearity to enforce a sharing of the computations, by disallowing duplication on certain subterms, using the separation binding syntax rather than a duplicable binding name. It ensures that the computation required to obtain the shape of a sum in the body of a substitution is not performed twice — as happens if the substitution is duplicated before this computation is performed.

Example 1.5. *Here is an example of a term where computation is shared among two possible copies of the same term. In the pure λ-calculus, we would write this term as $(\lambda b.(\lambda a.t)\ (b\ v))\ (\lambda c.u)$. But here, we want to ensure that the application of $\lambda a.t$ is fully reduced only after the function $\lambda c.u$ is applied to v. For that, we use sums:*

$$\begin{array}{ll} & (\lambda b.(\lambda a_1 + a_2.t)\ (b\ v))\ (\lambda c.u+u) \\ \longrightarrow_{\mathtt{B}} & ((\lambda a_1 + a_2.t)\ (b\ v))[b \leftarrow \lambda c.u+u] \\ \longrightarrow_{\mathtt{apr}} & (\lambda a_1 + a_2.t)\ (b\ v)[b \leftarrow \lambda c.u+u] \\ \longrightarrow_{\mathtt{apl}} & (\lambda a_1 + a_2.t)\ (b[b \leftarrow \lambda c.u+u]\ v) \\ \longrightarrow_{\mathtt{var}} & (\lambda a_1 + a_2.t)\ ((\lambda c.u+u)\ v) \\ \longrightarrow_{\mathtt{B}} & t[a_1 + a_2 \leftarrow (\lambda c.u+u)\ v] \\ \longrightarrow_{\mathtt{B}} & t[a_1 + a_2 \leftarrow (u+u)[c \leftarrow v]] \\ \longrightarrow_{\mathtt{dup}} & t[a_1 + a_2 \leftarrow (u_{[d/c]}+u)[d \leftarrow v][c \leftarrow v]] \\ \longrightarrow_{\mathtt{sul}} & t[a_1 + a_2 \leftarrow (u_{[d/c]}[d \leftarrow v]+u)[c \leftarrow v]] \\ \longrightarrow_{\mathtt{sur}} & t[a_1 + a_2 \leftarrow u_{[d/c]}[d \leftarrow v]+u[c \leftarrow v]] \\ \longrightarrow_{\mathtt{sep}} & t[a_1 \leftarrow u[c \leftarrow v]][a_2 \leftarrow u[c \leftarrow v]] \end{array}$$

where c occurrs only once in u. There are other possible reduction paths, but in all of them the duplication performed in the last step above comes after the application of the function $\lambda c.u+u$ to the term v, since the subterm $(y\ v)$ cannot be separated.

2 Reduction in the λr-calculus

The operational behaviour of the λr-calculus, defined by the set of rules shown in Figure 1, makes it a generalisation of the λs-calculus incorporating concepts from resource-conscious calculi such as the λ-calculus *with multiplicities* [Bou93] or its non-deterministic variant *with resources* [BCL99]. The idea in these two calculi is that in an application, a » *bag* « of terms is passed as argument rather than a simple term, so that one could write a term such as:

$$(\lambda x.x\ x)\,(u \mid v) \quad \text{which would reduce to} \quad (u\ v) \quad \text{or} \quad (v\ u)$$

In the deterministic version λm [BCL99], bags must be *uniform*, containing a unique term, repeated several times — in the example above, this is the case where $u = v$. Moreover, an argument u can be explicitly specified as having *infinite multiplicity*, which is written u^∞ and would correspond to an infinite bag $(u \mid \cdots \mid u)$. Notice that a bag may only appear directly as the argument in an application, so that there is a clear distinction between the level of bags and the level of terms.

In the λr-calculus, we replace bags of terms by terms where bags of terms can appear anywhere. The equivalent of a bag $(u \mid v)$ is built using the sum syntax, as $u + v$. The novelty is that sums do not only appear at applications, but anywhere in a term. However, this comes at a price, as reduction fails if no matching appears between the separation described by a compound binder and the sums used in the corresponding body. For this reason, λr lacks many good properties, at least in the untyped case.

2.1 Operational Properties of λr

The introduction of sums in the setting of explicit substitutions makes the situation in λr more complex than in λs, because of the subtle equilibrium required when sums and compound bindings are involved in a term, for it to reduce properly. In the case of terms not using sums and only basic names in binders, good properties are recovered since the system behaves exactly like λs — on such a simple term, it is therefore confluent, and β-reduction can be simulated stepwise by a translation into this subset, which preserves strong normalisation.

Deadlocks and resources. The rules sep and sum are the only two rules of λr involving the sum syntax and compound binders, but they can lead to cases where a term does not reduce to a proper normal form, even if it has only finite reduction sequences. This phenomenon, known as a *deadlock*, also happens in other calculi with resources such as λm [BCL99], when not enough resources are provided to a function, as arguments in an application.

In the λr-calculus, we have a deadlock if the sum syntax blocks the reduction of a term containing explicit substitutions, for example if the separation indicated by a compound binding $x + y$ is needed but the body of the substitution does not reduce to the shape of a sum, as in the term:

$$\lambda c.(\lambda a + b.a\ b)\,(c\ u) \longrightarrow^{\equiv}_{\lambda r} \lambda c.(a\ b)[a + b \leftarrow c\ u]$$

In this situation, the term is a normal form for the $\longrightarrow^{\equiv}_{\lambda\mathtt{r}}$ reduction system, but it contains an explicit substitution and there is no way to remove it either by pushing it furthing inside to the variables a and b or by erasing it.

Definition 2.1. *A $\lambda\mathtt{r}$-term t is said to be in a* deadlock situation *if there are explicit substitutions in t and there is no $\lambda\mathtt{r}$-term u such that $t \longrightarrow^{\equiv}_{\lambda\mathtt{r}} u$, and a term v is said to be* reducible *when there is no t in deadlock such that $v \longrightarrow^{\equiv *}_{\lambda\mathtt{r}} t$.*

The difficulty of defining $\lambda\mathtt{r}$-terms using sums in a non-trivial way is to manage the global balance of compound bindings and sums, so that enough resources are provided for separations not to lead to a deadlock. This is particularly difficult for applications: in $t\ u$, if t is not already an abstraction, we cannot know how many copies of u will be needed during reduction.

Simulation of λ. In a calculus such as $\lambda\mathtt{s}$, explicit substitutions can always be pushed to variables, each one separately from the others: this is the *full composition* result. However, deadlocks in $\lambda\mathtt{r}$ disallow any such result in general, which is the way of obtaining stepwise simulation of β-reduction in the calculus. A translation chosen for simulation must produce a well-balanced term, from some subset of $\lambda\mathtt{r}$ that enjoys full composition, and there is at least one, because as mentioned before, $\lambda\mathtt{s}$ is the fragment of $\lambda\mathtt{r}$ not using sums. Therefore, we consider the translation $[\![\cdot]\!]^{\lambda}_{\mathtt{s}}$ — defined as the identity on λ-terms, where variables in λ are interpreted as basic names — and prove the simulation result.

Proposition 2.2. *For any λ-terms t and u, if $t \longrightarrow_{\beta} u$ then $[\![t]\!]^{\lambda}_{\mathtt{s}} \longrightarrow^{\equiv}_{\lambda\mathtt{r}} [\![u]\!]^{\lambda}_{\mathtt{s}}$.*

Proof. The fragment of $\lambda\mathtt{r}$ where no sum and no compound binding is used in any term is exactly $\lambda\mathtt{s}$, and we know [KR11] that $[\![t]\!]^{\lambda}_{\mathtt{s}} \longrightarrow^{\equiv}_{\lambda\mathtt{s}} [\![u]\!]^{\lambda}_{\mathtt{s}}$, based on the full composition result in this setting. Using exactly the same rules in $\lambda\mathtt{r}$ provides the expected reduction sequence. □

It is thus possible to simulate the λ-calculus in $\lambda\mathtt{r}$, but restricting terms to the $\lambda\mathtt{s}$ fragment provides no information on the properties of terms using sums, so that it is of limited interest. Another possibility for simulation would be to use the other extreme in the spectrum, and translate an abstraction as follows:

$$[\![\lambda a.t]\!] \ = \ \lambda a_1 + \cdots + a_n.[\![t_{[a_1/a]\cdots[a_n/a]}]\!] \qquad \text{where } |t|_a = n$$

However, it would require to provide enough copies of the translation of arguments when translating an application, turning $[\![t\ u]\!]$ into $[\![t]\!]\ ([\![u]\!] + \cdots + [\![u]\!])$. If the translation is not in normal form, we cannot know how many $[\![u]\!]$ copies will be needed without reducing t. Worst, subterms inside t and u might interact, so that for example applications in t must match the binding of the correct abstraction in u. This is in general a difficult problem, related to the transformation called *weak linearisation* [AF05] in the λ-calculus. Finally, defining a more general translation allowing to share parts of a term is even more difficult, as it involves an analysis of what other parts of the term will be duplicated and in which order separation and duplication should happen, on a global level.

Note that in the other direction, projecting reduction of $\lambda\mathtt{r}$ into β-reduction is also complicated: one the side of applications, the sum $u+v$ can be encoded as the pair (u,v) through the usual encoding of pairs in the λ-calculus, but representing an abstraction $\lambda x+y.t$ requires to use several projections $\mathtt{fst}$ and $\mathtt{snd}$ to decompose the binding. The translation can be defined as the identity on variables, and:

$$\begin{array}{rcl@{\qquad}rcl} [\![t\ u]\!]^{\lambda}_{\mathtt{r}} & = & [\![t]\!]^{\lambda}_{\mathtt{r}}\,[\![u]\!]^{\lambda}_{\mathtt{r}} & [\![\lambda a.t]\!]^{\lambda}_{\mathtt{r}} & = & \lambda a.[\![a]\!]^{\lambda}_{\mathtt{r}} \\ [\![t+u]\!]^{\lambda}_{\mathtt{r}} & = & ([\![t]\!]^{\lambda}_{\mathtt{r}}, [\![u]\!]^{\lambda}_{\mathtt{r}}) & [\![\lambda x+y]\!]^{\lambda}_{\mathtt{r}} & = & \lambda a.[\![\lambda x.\lambda y.t]\!]^{\lambda}_{\mathtt{r}}\,(\mathtt{fst}\ a)\,(\mathtt{snd}\ a) \end{array}$$

but this is still problematic to project the $\mathtt{sep}$ rule, since it would require to move the projections $\mathtt{fst}$ and $\mathtt{snd}$ at the position in the term where separation happens in the reduction step we are projecting.

Confluence. Besides simulation of the standard β-reduction, one of the most important property expected from a λ-calculus is confluence, to ensure that when a normal form is reached by reduction of a term, it is unique. In general, this cannot hold in $\lambda\mathtt{r}$, because of deadlocks. Consider the following critical pair:

$$\begin{array}{ccc} & (a\ b)[a+b\leftarrow c+d][c+d\leftarrow u] & \\ \swarrow & & \searrow \\ (a\ b)[a\leftarrow c][b\leftarrow d][c+d\leftarrow u] & & (a\ b)[a+b\leftarrow (c+d)[c+d\leftarrow u]] \end{array}$$

Here, if u is in normal form and is not of the shape of a sum, then the substitution on $c+d$ on the right cannot be separated to be pushed in both substitutions on a and b, while it is already inside on the right. We end up with two different normal forms:

$$(c\ d)[c+d\leftarrow u] \ \neq\ (c[c+d\leftarrow u])\,(d[c+d\leftarrow u])$$

because of the two different positions where the explicit substitution was blocked during reduction. This leads to the conjecture that $\lambda\mathtt{r}$ is confluent when restricted to reducible terms, where such deadlocks never happen — proving it would require a further study of the different deadlock cases.

Sums and non-determinism. As defined, $\lambda\mathtt{r}$ is deterministic in the sense that the structure of sums cannot change, and the association of left and right terms in sums with names in bindings is fixed. Bindings must match the sum structure in a term, but this can be relaxed in a non-deterministic setting by using equations:

$$t+u\ \equiv\ u+t \qquad\qquad t+(u+v)\ \equiv\ (t+u)+v$$

so that the association between names and terms can be changed. The equation for commutativity in particular induces a non-deterministic reduction. Allowing such non-determinism would induce the possibility to define terms with a rich behaviour, such as a symmetric form of application:

$$v\,u \quad \longleftarrow^{*} \quad (x\ y)[x+y\leftarrow u+v] \quad \longrightarrow^{*} \quad u\,v$$

or internal choice, with $u\oplus v = x[x+y\leftarrow u+v]$.

$$\texttt{var}\,\frac{}{a:\{\emptyset\}\triangleright A\vdash a:A}\qquad \texttt{sub}\,\frac{\Gamma,x:\{\Delta\vdash u:A\}\triangleright A\vdash t:B}{\Gamma,\Delta\vdash t[x\leftarrow u]:B}$$

$$\texttt{lam}\,\frac{\Gamma,x:A\vdash t:B}{\Gamma\vdash\lambda x.t:A\to B}\qquad \texttt{app}\,\frac{\Gamma\vdash t:(\{\Delta\vdash u:A\}\triangleright A)\to B}{\Gamma,\Delta\vdash t\,u:B}$$

$$\texttt{rem}\,\frac{\Gamma\vdash t:B}{\Gamma,a:\{\mathscr{U}\}\triangleright A\vdash t:B}\qquad \texttt{cpy}\,\frac{\Gamma,a:\{\mathscr{U}^\infty\}\triangleright A,b:\{\mathscr{U}^\infty\}\triangleright A\vdash t_{[b/a]}:B}{\Gamma,a:\{\mathscr{U}^\infty\}\triangleright A\vdash t:B}$$

$$\texttt{sep}\,\frac{\Gamma,x:\{\mathscr{U}_1\}\triangleright A,y:\{\mathscr{U}_2\}\triangleright A\vdash t:B}{\Gamma,x+y:\{\mathscr{U}_1+\mathscr{U}_2\}\triangleright A\vdash t:B}$$

Figure 2: Nested type system Nr for the λr-calculus

2.2 Nested Typing for λr

The type system Nr for the λr-calculus is based on a variant of the JD proof system for intuitionistic logic[1], and therefore involves nested typing structures, as all other nested type systems in *» natural deduction style «* described in Chapter 5. The basic notions are defined as in other systems:

$$\mathscr{U}\ ::=\ x:\phi\vdash\mathscr{U}\ \mid\ t:T\ \mid\ \emptyset\qquad T\ ::=\ A\ \mid\ \phi\to T\qquad \phi\ ::=\ \{\mathscr{U}\}\triangleright A$$

We denote *plain types*, which are intuitionistic formulas without the truth unit, and *conditional types* — corresponding to T in the grammar above — the same way, for the sake of simplicity. Notice that a usual typing hypothesis $x:A$, as used in type systems based on natural deduction, is now represented by the more complicated syntax $x:\{\emptyset\}\triangleright A$, as there can be conditions attached to typing assumptions. We can keep the standard notation $x:A$ in this setting, since A could be a conditional type and can thus contain a nested judgement.

Remark 2.3. *The binding names used in typing assumptions are in general any valid name from the calculus, so that it includes compound bindings of the shape $x+y$, or basic binding names such as a.*

The typing rules of the Nr system for λr are shown in Figure 2. It is similar to other type systems for more standard calculi, as described in Chapter 5, but has two distinct rules for the duplication of variables in the typing environment, depending on the cause of the duplication — either the multiple use of a basic name in a term, or a compound binding matching a sum subterm. The typing rule for separation is slightly more complex than more usual typing rules, as it uses a notation indicating that a judgement has a certain shape, ready to be separated.

[1] The particular system used here is JDr ∪ {u}, the variant of JD where the switch rule is built-in and the compound cut rule u are used, and where the limited contraction is added to the system.

Definition 2.4. *Any uniform typing structure $\mathscr{U} = \Gamma, \Delta \vdash u_1 + u_2 : A$ of $\lambda\mathtt{r}$ is said to be* separable *in the two structures $\mathscr{U}_1 = \Gamma \vdash u_1 : A$ and $\mathscr{U}_2 = \Delta \vdash u_2 : A$, which will be denoted by $\mathscr{U} \simeq \mathscr{U}_1 + \mathscr{U}_2$ — and by convention we also have $\emptyset \simeq \emptyset + \emptyset$.*

This separation relation is implicitly used in the Nr type system, to ensure that a judgement can be separated along a sum when applying the sep rule. It ensures that compound bindings will be separated only when the body of the corresponding substitutions have the shape of a sum — or when they are used in an abstraction.

The meaning of the sep typing rule introduced for $\lambda\mathtt{r}$ is that when a typing assumption is labelled with a compound name $x + y$ and contains a judgement to type a sum $u+v$, this assumption can be splitted along the sum, and each judgement on u and v obtained by separation is associated to the corresponding name, that is x and y respectively.

Example 2.5. *Below is shown an example of a partial derivation in* Nr *illustrating the use of the* sep *typing rule, which requires to split a judgement on $u + v$:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, a : \{\, \Delta \vdash u : A \,\} \triangleright A \vdash t : A \to B}{\Gamma, a : \{\, \Delta \vdash u : A \,\} \triangleright A \vdash t : (\{\, b : A \vdash b : A \,\} \triangleright A) \to B}\,\mathtt{var}}{\Gamma, a : \{\, \Delta \vdash u : A \,\} \triangleright A \vdash t : (\{\, b : \{\, \Psi \vdash v : A \,\} \triangleright A \vdash b : A \,\} \triangleright A) \to B}\,\mathtt{app}}{\Gamma, a : \{\, \Delta \vdash u : A \,\} \triangleright A, b : \{\, \Psi \vdash v : A \,\} \triangleright A \vdash t\ b : B}\,\mathtt{sep}}{\Gamma, a + b : \{\, \Delta, \Psi \vdash u + v : A \,\} \triangleright A \vdash t\ b : B}\,\mathtt{sub}$$
$$\Gamma, \Delta, \Psi \vdash (t\ b)[a + b \leftarrow u + v] : B$$

The cpy rule is a variant of the dup rule from Chapter 5, where the typing structure being duplicated is verified: the rule can only be applied if all judgements inside this structure are duplicable as well.

Definition 2.6. *Any uniform typing structure $\mathscr{U}$ of $\lambda\mathtt{r}$ is said to be* duplicable*, and is then denoted by $\mathscr{U}^\infty$, if and only if any typing assumption anywhere inside $\mathscr{U}$ is of the shape $a : \{\, \mathscr{U}_a \,\} \triangleright A$ — that is, if none of them has a compound binding.*

The Nr type system has properties similar to all the other nested type systems, as presented in Chapter 5. In particular, the typing process is terminating, as can be shown by defining an appropriate measure on typing structures, following the same scheme as in Nx and Ns. Although the type assigned to a $\lambda\mathtt{r}$-term is unique, up to renaming, the Nr system suffers from the same mismatch between terms and derivations as the others: there are many possible typing derivations for one given term, because of trivial permutations that are not visible in the term structure.

Remark 2.7. *Here, we have several ways of interpreting a given proof as the typing derivation of a $\lambda\mathtt{r}$-term, because of the two possible interpretations of the contraction rule. It can be used as a separation of a sum or as a plain duplication if the subproofs of the two copies of the contracted structure are the same. A canonical way of considering a proof is to turn all contractions into separations, producing a unique $\lambda\mathtt{r}$-term with no duplication, which is a » linearisation « of any other term interpreting this proof.*

The correspondence between the $\lambda\mathtt{r}$-calculus and the $\mathsf{JDs}\cup\{\mathtt{u}\}$ system lies in the matching between reduction rules in $\lambda\mathtt{r}$ and rewriting steps used in the detour elimination procedure used in $\mathsf{JDs}\cup\{\mathtt{u}\}$. As in the systems described in Chapter 5, we can list the reduction rules defining $\longrightarrow_{\lambda\mathtt{r}}$ and observe how the rewriting on a term corresponds to a rewriting on its typing derivation. Most cases are exactly the same as for derivations of $\mathtt{Ns}$ typing $\lambda\mathtt{s}$-terms, since most rules of $\lambda\mathtt{r}$ are also rules of $\lambda\mathtt{s}$. Therefore, we describe now only the cases involving the contraction:

1. The reduction $t[a \leftarrow u] \longrightarrow_{\mathtt{dup}} t_{[b/a]}[a \leftarrow u][b \leftarrow u]$ reflects a permutation of a cut above a contraction, so that if we consider the following derivation:

$$\mathtt{cpy}\,\dfrac{\xi\{\Gamma, x : \{\mathscr{U}^\infty\} \triangleright A, y : \{\mathscr{U}^\infty\} \triangleright A \vdash t_{[y/x]} : B\}}{\xi\{\Gamma, x : \{\mathscr{U}^\infty\} \triangleright A \vdash t : B\}}$$
$$\mathscr{D}\,\|$$
$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, x : \{\Delta \vdash u : A\} \triangleright A \vdash t : B\}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u] : B\}}$$

   we observe that we can duplicate the environment Δ first, because all assumptions inside it are on basic names — $\mathscr{D}$ cannot contain any $\mathtt{sep}$ instance, since such an instance could permute below $\mathtt{sub}$ —, and then use twice the $\mathtt{sub}$ rule, filling the gaps with copies of the $\mathscr{D}$ subderivation:

$$\dfrac{\xi\{\Gamma, x : \{\mathscr{U}^\infty\} \triangleright A, y : \{\mathscr{U}^\infty\} \triangleright A \vdash t_{[y/x]} : B\}}{\mathscr{D}\,\|}$$
$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, x : \{\Delta \vdash u : A\} \triangleright A, y : \{\mathscr{U}^\infty\} \triangleright A \vdash t_{[y/x]} : B\}}{\xi\{\Gamma, \Delta, y : \{\mathscr{U}^\infty\} \triangleright A \vdash t_{[y/x]}[x \leftarrow u] : B\}}$$
$$\mathscr{D}\,\|$$
$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, \Delta, y : \{\Delta \vdash u : A\} \triangleright A \vdash t_{[y/x]}[x \leftarrow u] : B\}}{\mathtt{cpy}^*\,\dfrac{\xi\{\Gamma, \Delta, \Delta \vdash t_{[y/x]}[x \leftarrow u][y \leftarrow u] : B\}}{\xi\{\Gamma, \Delta \vdash t_{[y/x]}[x \leftarrow u][y \leftarrow u] : B\}}}$$

The other principal case of contraction, where a typing structure is separated along a sum, is simpler, but there are two possibilities: either the duplicated assumption is exactly the one introduced by the cut, or it is inside some larger assumption being duplicated. We consider first the case where the contraction is duplicating exactly the typing structure introduced with the cut.

2. The reduction rule $t[x+y \leftarrow u+v] \longrightarrow_{\mathtt{sep}} t[x \leftarrow u][y \leftarrow v]$ is the reflection of the other main permutation of a cut above a contraction instance, so that if we consider the following derivation:

$$\mathtt{sep}\,\dfrac{\xi\{\Gamma, x : \{\Delta \vdash u : A\} \triangleright A, y : \{\Psi \vdash v : A\} \triangleright A \vdash t : B\}}{\mathtt{sub}\,\dfrac{\xi\{\Gamma, x + y : \{\Delta, \Psi \vdash u + v : A\} \triangleright A \vdash t : B\}}{\xi\{\Gamma, \Delta \vdash t[x + y \leftarrow u + v] : B\}}}$$

where no rule can be blocked above the cut — because of the shape of the body of the substitution —, we get the following derivation:

$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, x:\{\Delta \vdash u:A\}\triangleright A, y:\{\Psi \vdash v:A\}\triangleright A \vdash t:B\}}{\mathtt{sub}\,\dfrac{\xi\{\Gamma,\Delta, y:\{\Psi \vdash v:A\}\triangleright A \vdash t[x\leftarrow u]:B\}}{\xi\{\Gamma,\Delta,\Psi \vdash t[x\leftarrow u][y\leftarrow v]:B\}}}$$

3. The reduction rule $(t+v)[x\leftarrow u] \longrightarrow_{\mathtt{sul}} t[x\leftarrow u]+v$ shows one of the last two cases involving a contraction, where the cut goes through a contraction that separates the inner contexts, and goes only on the left, so that:

$$\mathtt{sep}\,\dfrac{\xi\{\Gamma, y:\{\Delta, x:\{\mathscr{U}\}\triangleright A \vdash t:B\}\triangleright C, z:\{\Psi \vdash v:B\}\triangleright C \vdash p:D\}}{\xi\{\Gamma, y+z:\{\Delta,\Psi, x:\{\mathscr{U}\}\triangleright A \vdash t+v:B\}\triangleright C \vdash p:D\}}$$
$$\mathscr{D}\Vert$$
$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, y+z:\{\Delta,\Psi, x:\{\Sigma \vdash u:A\}\triangleright A \vdash t+v:B\}\triangleright C \vdash p:D\}}{\xi\{\Gamma, y+z:\{\Delta,\Psi,\Sigma \vdash (t+v)[x\leftarrow u]:B\}\triangleright C \vdash p:D\}}$$

can be turned into the following derivation:

$$\xi\{\Gamma, y:\{\Delta, x:\{\mathscr{U}\}\triangleright A \vdash t:B\}\triangleright C, z:\{\Psi \vdash v:B\}\triangleright C \vdash p:D\}$$
$$\mathscr{D}\Vert$$
$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, y:\{\Delta, x:\{\Sigma \vdash u:A\}\triangleright A \vdash t:B\}\triangleright C, z:\{\Psi \vdash v:B\}\triangleright C \vdash p:D\}}{\mathtt{sep}\,\dfrac{\xi\{\Gamma, y:\{\Delta,\Sigma \vdash t[x\leftarrow u]:B\}\triangleright C, z:\{\Psi \vdash v:B\}\triangleright C \vdash p:D\}}{\xi\{\Gamma, y+z:\{\Delta,\Psi,\Sigma \vdash t[x\leftarrow u]+v:B\}\triangleright C \vdash p:D\}}}$$

As in other nested systems, the composition rule $\mathtt{cmp}$ and the equation $\equiv_e$ that is introduced with it to handle the exchange of independent substitutions correspond to trivial permutations. The correspondence between reduction in $\lambda\mathtt{r}$ and detour elimination in $\mathsf{JDs}\cup\{\mathsf{u}\}$ described in this case analysis and in Chapter 5 leads us to the conclusion that if a $\lambda\mathtt{r}$-term admits a typing derivation, there exists a strategy for reducing it into a normal form.

Theorem 2.8 (Subject reduction)**.** *If* $\Gamma \vdash t:A$ *and* $t \longrightarrow^{\equiv}_{\lambda\mathtt{r}} u$ *then* $\Gamma \vdash u:A$*.*

Proof. This result can be obtained by inspection of the rewriting cases shown above, since each reduction rule in the $\lambda\mathtt{r}$-calculus corresponds to a case of moving a cut above another rule instance, or creating a cut in the case of $\mathtt{B}$. □

Theorem 2.9 (Normalisation)**.** *For any* $\lambda\mathtt{r}$*-term* t*, if* $\Gamma \vdash t:A$ *there is a* $\lambda\mathtt{r}$*-term* u *such that we have* $t \longrightarrow^{\equiv *}_{\lambda\mathtt{r}} u$ *and* u *is a normal form.*

Proof. This immediately follows from the termination of the procedure for detour elimination in Chapter 4 and Theorem 2.8, since any reduction step corresponds to a detour elimination step applied in the typing derivation. Moreover, by definition, when a typing derivation is normal, we know that the corresponding term contains no explicit substitution and no β-redex. □

Since the $\lambda\mathtt{r}$-calculus lacks good properties on the untyped level, this result is rather weak, but we can conjecture that typing ensures reducibility by its checking of the balance of sums, and this could imply also confluence.

3 Switch and Communication

The switch rule is one of the most particular features of deep inference formalisms, and it is essentially an expression of the interplay between subformulas at different depth in a structure. As a fine-grained rule performing an operation that cannot be perceived in shallow formalisms, interpreting in terms of computation its meaning, through the typing methodology, is a way of obtaining more insights on the nature of the compound operations it is involved in.

3.1 The Decomposition of Context Splitting

In shallow formalisms such as the sequent calculus or natural deduction, the rules involving branching must deal, besides the » *principal formula* «, with some context that needs to be splitted or duplicated, and distributed among both premises. In a system in the calculus of structures, this is usually separated from the rule, as the basic mechanism of distribution is built in the switch rule, as shown below:

$$\rightarrow_{\mathsf{e}} \frac{C_1, \cdots, C_n \vdash A \qquad \Gamma \vdash A \rightarrow B}{\Gamma, C_1, \cdots, C_n \vdash B} \quad \longrightarrow \quad \mathsf{e}\,\frac{\mathsf{si}\,\frac{\mathsf{si}\,\frac{\mathsf{si}\,\frac{\Gamma \Rightarrow (((C_1 \Rightarrow \cdots \Rightarrow C_n \Rightarrow A) \Rightarrow A) \rightarrow B)}{\cdots}}{\Gamma \Rightarrow C_1 \Rightarrow \cdots \Rightarrow (((C_n \Rightarrow A) \Rightarrow A) \rightarrow B)}}{\Gamma \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_n \Rightarrow ((A \Rightarrow A) \rightarrow B)}}{\Gamma \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_n \Rightarrow B}$$

Here, the implication elimination rule from the standard natural deduction system NJ is translation into the JD system not only using the » *corresponding* « rule e, but also using several switch instances. The purpose of these instances is to distribute all of the C_i structures to the translation of the left branch, which is nested in the translation of the right branch. The JD derivation above does exactly the same as the NJ instance, but it makes each step of the distribution process *explicit*.

Providing a computational interpretation of the switch rule in JD, based on the nested typing methodology described in Chapter 5, requires to make distribution explicit in a λ-calculus as well. In this setting, a rule instance that pushes a typing assumption inside a subjudgement is exposed when a cut creates this assumption, and must therefore be pushed inside during its elimination. On the side of terms, the nesting of judgements is represented by binary operations such as application, where the inner judgement is associated to the argument, and the operation that a switch performs is thus a » *jump* « from a subterm into its argument:

$$(\lambda y.(\mathtt{jump}.t)[x \leftarrow u])\,(\mathtt{here}.v) \quad \longrightarrow \quad (\lambda y.t)\,(v[x \leftarrow u])$$

and we will use a pair of operators following the `jump/here` scheme above to give a term annotation to the switch rule — note that one rule will thus type two operators at a time, taken as a pair. Also, as in the example, a substitution can jump out from deep inside a term, creating » *backwards references* « that describe a *global* form of communication, where the subterms of an application are distinct processes, much like in the π-calculus [Mil99].

3.2 Syntax of the λc-calculus

As done for resources in λr, the syntax we use to introduce communication in the λc-calculus is a simple extension of the syntax of the λs-calculus [KR11]. It relies on the use of a pair of operators for input and output, used to communicate explicit substitutions, considered as resources, from one subterm to another subterm. This is based on the use of *channels*, so that we need to assume given a countable set of channels denoted by greek letters such as α, κ or γ. As usual, variables are denoted by letters such as x, y and z, and terms by t, u or v.

Definition 3.1. *The language of terms of the λc-calculus is defined as:*

$$t, u ::= x \mid \lambda x.t \mid t\ u \mid t[x \leftarrow u] \mid \alpha x.t \mid \overline{\alpha} x.t$$

The notations used for the input construct αx and the output construct $\overline{\alpha} x$ are meant to emphasize the idea of communicating along some channel α, as done in the π-calculus [Mil99]. As usual with communication primitives, input is binding, so that if we write $\alpha x.t$ then x is bound in t. This can be observed in the definition of free and bound variables.

Definition 3.2. *Given a λc-term t, the set* $\mathrm{bv}(t)$ *of its* bound variables *is:*

$$\begin{array}{rclcrcl} \mathrm{bv}(x) & = & \emptyset & & \mathrm{bv}(t\ u) & = & \mathrm{bv}(t) \cup \mathrm{bv}(u) \\ \mathrm{bv}(\lambda x.t) & = & \mathrm{bv}(t) \cup \{x\} & & \mathrm{bv}(t[x \leftarrow u]) & = & \mathrm{bv}(t) \cup \{x\} \cup \mathrm{bv}(u) \\ \mathrm{bv}(\alpha x.t) & = & \mathrm{bv}(t) \cup \{x\} & & \mathrm{bv}(\overline{\alpha} x.t) & = & \mathrm{bv}(t) \end{array}$$

while the set $\mathrm{fv}(t)$ *of its* free variables *is:*

$$\begin{array}{rclcrcl} \mathrm{fv}(x) & = & x & & \mathrm{fv}(t\ u) & = & \mathrm{fv}(t) \cup \mathrm{fv}(u) \\ \mathrm{fv}(\lambda x.t) & = & \mathrm{fv}(t) \setminus \{x\} & & \mathrm{fv}(t[x \leftarrow u]) & = & (\mathrm{fv}(t) \setminus \{x\}) \cup \mathrm{fv}(u) \\ \mathrm{fv}(\alpha x.t) & = & \mathrm{fv}(t) \setminus \{x\} & & \mathrm{fv}(\overline{\alpha} x.t) & = & \mathrm{fv}(t) \cup \{x\} \end{array}$$

It is important to notice that the syntax of λc contains no operator to bind the names of channels: if a channel α is used for an output in a subterm, then the other subterm that needs a resource passed through α must use an input on α. Handling channel names is therefore a *global* problem, and the names of the channels matter when composing terms — we can say that channel names are part of the *interface* of a term in λc, as in a process algebra. The basic syntax of λc-terms is simple, but we need to enforce one condition on the way they use variables.

Definition 3.3. *A λc-term t is said to be* well-formed *if and only if in any subterm of the shape $v\ u$ in t, we have* $\mathrm{fv}(u) = \emptyset$.

In the following, we only consider well-formed terms: the condition is slightly surprising, but it corresponds to the fact that the arguments are seen as separate subprocesses in an application. All the resources they need must be passed through communication, or given as arguments after reduction of the β-redex. There is no condition on the use of channels, and one channel name may appear several times and be used for input and output of different resources — as a consequence, the reduction system in λc will not be deterministic.

We use here the same notations as in $\lambda\mathtt{s}$, and α-conversion is used implicitly to rename bound variables, so that all $\lambda\mathtt{c}$-terms we consider must follow Barendregt's convention. We also write $|t|_x$ for the number of free occurrences of x in a term t, and $t_{[y/x]}$ for the non-deterministic replacement in t of exactly one occurrence of the variable x by another variable y. Notice that the syntactic condition *(2)* above allow to rename a channel, if it appears twice in a term, since this term cannot be composed with another term using this channel. Finally, the implicit substitution $\{u/x\}$ cannot be defined directly as in calculi such as $\lambda\mathtt{s}$ which are *» less explicit «*, only the renaming of variables $\{y/x\}$ is defined, as usual. Furthermore, we need some meta-notations.

Definition 3.4. *Given a list $\phi = \{x_1, \cdots, x_n\}$ of variables, the meta-notations $\overline{\alpha_i}\phi.t$ and $\alpha_i\phi.t$ represent of the terms $\overline{\alpha_1}x_1.\cdots.\overline{\alpha_n}x_n.t$ and $\alpha_1x_1.\cdots.\alpha_nx_n.t$ respectively, and t if ϕ is empty, where each α_i is a different channel.*

Threads hierarchy. The separation of arguments from the body of the function applied on them, made clear by the requirement that the arguments must be closed subterms, creates a situation where distinct *» threads «* are identified in a term. In this view, an application $t\ u$ is composed of two threads t and u, where the function body t can be called the *main thread*, while the argument u is an auxiliary thread — that might actually not be used, if t discards this argument. The thread t can in turn contain other applications, each of them with a main and an auxiliary thread, so that there is a hierarchy among threads in a term:

$$(t\ u) \quad \textit{is similar to} \quad (t \parallel u) \quad \textit{with some ordering } t \succ u$$

In this setting, applications play a particular role, since they form the structure of a $\lambda\mathtt{c}$-term, in terms of the communication required between threads. However, the explicit substitution obtained by reducing a β-redex induces a different status of its body: in $t[x \leftarrow u]$ the thread u is partially integrated into t, since they can share variables and thus have access to the same pool of resources. But here again there is a hierarchy where t is the main thread and u an auxiliary subterm. Because of this separation, we will need to define subclasses of normal contexts $C[-]$, where the distinction between threads is internalised.

Definition 3.5. *The* application contexts *and* spine contexts *of $\lambda\mathtt{c}$ are the classes of contexts denoted by $\pi\{-\}$ and $\vec{\pi}\{-\}$ respectively, and defined as:*

$$\pi ::= - \mid \lambda x.\pi \mid \pi[x \leftarrow u] \mid \alpha x.\pi \mid \overline{\alpha}x.\pi \qquad \vec{\pi} ::= \pi \mid \pi\{\vec{\pi}\,u\}$$

An application context $\pi\{-\}$ corresponds to a $\lambda\mathtt{c}$-term where the hole is located within the outermost main thread, as defined by an application. Then, any context $\vec{\pi}\{-\}$ represents the spine of a term, as iteration of application contexts. In such a context, the hole can be located anywhere on the left of any application.

Reductions rules. The set of rules defining the reduction system $\longrightarrow_{\lambda\mathtt{c}}$ for the $\lambda\mathtt{c}$-calculus is given in Figure 3. A large part of the rules are those of $\lambda\mathtt{s}$, but the particular approach to subterms in applications simplifies the way substitutions are pushed into applications. Moreover, some new rules are introduced to handle the communication mechanisms:

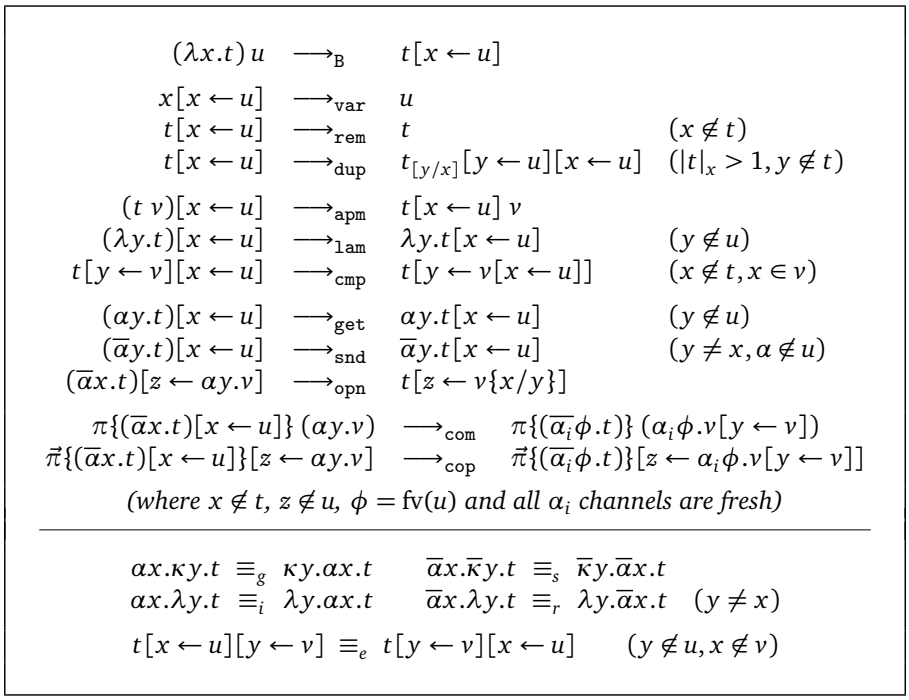


Figure 3: Reduction rules and equations of the λc-calculus

- The `get` rule simply pushes a substitution under an input operator, just as it would be pushed under an abstraction.

- The `snd` rule pushes a substitution under an output operator, but this is only valid when the output is not moving this substitution, and when the body of the substitution does not contain the matching input on the channel used.

- The `opn` rule removes a pair of matching input and output that have become useless by replacing the variable bound by the input in the substitution by the variable moved through the output — so that the body of the substitution contains one more free variable.

- The `com` rule performs the communication of one substitution over a channel, from the main thread to its argument in an application, using the name from the input for the resulting substitution, and erasing the channel — and fresh channels must also be introduced to prepare the moving of substitutions on the free variables of the substitution.

- The `cop` rule performs the same communication as `com`, but from the main thread to the body of another substitution, and also introduces fresh channels for future communications.

The opn rule may seem disturbing, since it erases a pair of valid communication operators, but it should be noticed that there are two rules performing composition of substitutions: the standard cmp and the new cop. As in the case handled by opn the standard rule can be applied on the result, this can be considered as a » *garbage collection* « of communication channels.

The heart of the reduction system of λc are the com and cop rules, that allow to create the flow of resources from the main thread of a term to its arguments and the subterms inside substitutions. Note that com is the only link between a function and its argument before the corresponding β-redex is triggered: it introduces some necessary substitutions into the closed argument and leaves the subterm closed by introducing inputs for the new free variables.

The reduction system comes with the standard equation $\equiv_e$ for the exchange of independent explicit substitutions, but also with equations allowing to exchange two inputs, or two outputs, as they are always independent — that would not be the case for an input exchanged with an output. The two last equations $\equiv_i$ and $\equiv_r$ allow to exchange an abstraction with inputs and outputs, in the cases where they are independent. This avoids the potential problem of β-redexes being blocked by communication operators. The congruence defined by these equations is integrated as usual to the reduction system, and the resulting relation is called $\longrightarrow^{\equiv}_{\lambda\mathtt{c}}$. Finally, we can easily prove that this reduction preserves well-formedness.

Proposition 3.6. *If t is a well-formed λc-term and we have $t \longrightarrow^{\equiv}_{\lambda\mathtt{c}} u$ then u is also a well-formed λc-term.*

Proof. First, the equations shown in Figure 3 preserve well-formedness since none of them induces the creation of new free variables, and they preserve bindings. In the reduction rules, there is also no creation of new free variables, and binders are always either erased with all of their scope or replaced by another kind of binder on the same name — in the case of opn, the renaming ensures that the y is not left free, since if x is free before then x is not a new free variable after application. □

Example 3.7. *Here is an example of a reduction of term performing a communication across an application and also to a subterm inside an explicit substitution:*

$$\begin{array}{ll}
 & \lambda w.((\lambda z.\overline{\alpha}x.t)\,(\alpha y.(\overline{\kappa}y.v)[n \leftarrow \kappa x.x]))[x \leftarrow w] \\
\longrightarrow_{\mathtt{apm}} & \lambda w.(\lambda z.\overline{\alpha}x.t)[x \leftarrow w]\,(\alpha y.(\overline{\kappa}y.v)[n \leftarrow \kappa x.x]) \\
\longrightarrow_{\mathtt{lam}} & \lambda w.(\lambda z.(\overline{\alpha}x.t)[x \leftarrow w])\,(\alpha y.(\overline{\kappa}y.v)[n \leftarrow \kappa x.x]) \\
\longrightarrow_{\mathtt{com}} & \lambda w.(\lambda z.\overline{\gamma}w.t)\,(\gamma w.(\overline{\kappa}y.v)[n \leftarrow \kappa x.x][y \leftarrow w]) \\
\equiv_e & \lambda w.(\lambda z.\overline{\gamma}w.t)\,(\gamma w.(\overline{\kappa}y.v)[y \leftarrow w][n \leftarrow \kappa x.x]) \\
\longrightarrow_{\mathtt{cop}} & \lambda w.(\lambda z.\overline{\gamma}w.t)\,(\gamma w.(\overline{\upsilon}w.v)[n \leftarrow \upsilon w.x[x \leftarrow w]]) \\
\longrightarrow_{\mathtt{opn}} & \lambda w.(\lambda z.\overline{\gamma}w.t)\,(\gamma w.v[n \leftarrow x[x \leftarrow w]]) \\
\longrightarrow_{\mathtt{var}} & \lambda w.(\lambda z.\overline{\gamma}w.t)\,(\gamma w.v[n \leftarrow w])
\end{array}$$

Notice how the communication rules leave a trail in the term of outputs and rebinding of w by inputs, a part of it being collected in the end by the opn *rule. There are four threads coming into play here, one per substitution, a main thread in the application and an argument thread. Also, the channels γ and υ introduced during reduction are fresh — we assume given a mechanism to retrieve fresh channel names.*

4 Reduction in the λc-calculus

As mentioned before, the reduction system of λc can be highly non-deterministic, because of the use of channels. If input and output operators are used in a term at proper locations, each using a different channel, then a given explicit substitution can be carried out, but if the communication structure is not properly built in this term, the substitution might not reach its destinations.

The most » *natural* « location for a communication operator is at toplevel in the left subterm of an application for outputs, and at toplevel in the right subterm of an application for inputs, since this is where they are needed for the com reduction rule to apply. In these positions, they are nothing more than indications of which names are free in the argument, used to guide substitutions as *director strings* [KS88], as it has been used in the λ-calculus to obtain representations of terms without names or indices, and study evaluation strategies [FMS05]:

$$(t\ v)^{\sigma}[x \leftarrow u] \longrightarrow t[x \leftarrow u]\ v \qquad (\sigma \text{ associates } x \text{ to the left})$$
$$(t\ v)^{\sigma}[x \leftarrow u] \longrightarrow t\ v[x \leftarrow u] \qquad (\sigma \text{ associates } x \text{ to the right})$$

In the λc-calculus, these directors σ have been internalised in the syntax of terms and allowed not only at the natural locations, but anywhere in a term, even in some subterm disjoint from the subterm where they will be needed. As a consequence, the operators implementing directors must be moved to a position where the rules for communication can use them: in λc, *the communication infrastructure of a term can be dynamically created during reduction*. The price for this rich behaviour is the lack of good properties in comparison with the λs-calculus it is based on, and the difficulty to reason on its complex reduction dynamics.

4.1 Operational Properties of λc

The complexity of the reduction dynamics of λc, induced by the essential use made of communication rules and channels, and the possibility for an explicit substitution to jump from one subterm out to another subterm, makes it complicated to prove any general result in this setting. But as we will see, some λc-terms behave better than others, and might allow to prove limited results.

Deadlocks and synchronisation. In the communication rules com and cop, as well as in opn, the reduction relies on the particular shape of both the term under the substitution and the body of the substitution. As in λr, this requirement made on several independent constructions in the term can lead to *deadlock* situations, if one part of the term does not reduce to the expected shape. Consider the term:

$$\lambda z.(\overline{\alpha}w.t)[x \leftarrow z\ (\alpha y.u)]$$

Here, because the input on channel α is located under an application, the opn rule cannot be applied and therefore the substitution cannot be pushed inside t. Notice that plugging a term inside a context allows to » *unlock* « a deadlock situation: for example, applying the term above to the identity $\lambda x.x$, allows to pop the input on α out of the application, and use the opn rule. Also the communication rules com and cop can produce deadlock situations.

The creation of deadlocks by reduction rules handling communication between subterms denotes the presence of *concurrency* in the λc-calculus. Indeed, subterms in two different threads can be thought of as being executed in parallel, when both have redexes to reduce, but deadlocks represent situations where one thread waits on the other. This is similar to the situation in the π-calculus:

$$\overline{x}\langle y\rangle.P \parallel x(z).Q \quad \longrightarrow \quad P \parallel Q\{y/z\}$$

where communication can happen only when both processes in parallel have the right prefix, involving channel x. For example, if some process reduces into $x(z).Q$ then the communication cannot happen before it reaches this form — and it might rely on the replacement to reduce further. Following this analogy[2], we have in λc:

$$(\overline{\alpha}y.p)[y \leftarrow u]\,(\alpha z.q) \qquad \text{corresponds to} \qquad \overline{x}\langle y\rangle.P \parallel x(z).Q$$

where we can see that the resource being transmitted is more than just a name y, since λ-calculi are *higher-order* — in the π-calculus, this can be done by extending the language [San93]. As in λr, we can make a distinction between terms blocked on communication and those where communication always goes through.

Definition 4.1. *A λc-term t is said to be in a* deadlock situation *if there are explicit substitutions in t and there is no λc-term u such that $t \longrightarrow^{\equiv}_{\lambda c} u$, and a term v is said to be* reducible *when there is no t in deadlock such that $v \longrightarrow^{\equiv *}_{\lambda c} t$.*

Again, avoiding deadlocks is a global problem, since it requires a balance among different inputs and outputs on various channels, where channel names are used in a cautious way to avoid any clash, and misrouting of the resources. When defining λc-terms, we must always keep in mind that they will have to *sychronise* at some point with other terms to receive resources.

Simulation of λ. Although λc is based on the λs-calculus, which can simulate the reduction of the standard λ-calculus, the problem of deadlocks can prevent an explicit substitution to be carried out completely. Moreover, a pure λ-term is a valid λc-term only if all the subterms used as arguments in an application are closed, so that a translation cannot be defined as easily as in λr. However, there exists one translation of λ-terms into λc-terms allowing a kind of simulation of β-reduction — the idea is to place inputs and outputs at their » *natural* « positions, much like weakenings and contractions in the λlxr-calculus [KL07].

Definition 4.2. *The translation $[\![\cdot]\!]^{\lambda}_{c}$ from pure λ-terms into λc-terms is defined as:*

$$\begin{aligned} [\![x]\!]^{\lambda}_{c} &= x \\ [\![\lambda x.t]\!]^{\lambda}_{c} &= \lambda x.[\![t]\!]^{\lambda}_{c} \\ [\![t\ u]\!]^{\lambda}_{c} &= (\overline{\alpha_i}\phi.t)\,(\alpha_i\phi.u) \qquad \textit{where } \phi = \mathrm{fv}(u)\textit{, and all } \alpha_i \textit{ are fresh} \end{aligned}$$

Note that during translation, a given channel name is used only once, since we pick fresh channels when translating an application — that is, a channel introduced there should be unused in any other part of the whole resulting term.

[2] In this analogy, the terms in an application in λc are considered as parallel, since as we mentioned the argument thread is independent, in the sense that it is a closed term, although there is a hierarchy.

With this translation, all the communication operators required to carry out any substitution are located at the positions where they will be needed. This allows to prove a variant of the full composition result.

Lemma 4.3. *For any λ-terms t and u, we have* $[\![t]\!]^{\lambda}_{\mathtt{c}}[x \leftarrow [\![u]\!]^{\lambda}_{\mathtt{c}}] \longrightarrow^{\equiv *}_{\mathtt{c}} [\![t\{u/x\}]\!]^{\lambda}_{\mathtt{c}}$.

Proof. We proceed by structural induction on t. First, if x does not appear in t then it cannot appear in $[\![t]\!]^{\lambda}_{\mathtt{c}}$, we can use the `rem` rule and we are done. If t is x, then we conclude using `var`, and if t is $\lambda y.v$ then we use the induction hypothesis. If t is $p\ q$, then $[\![t]\!]^{\lambda}_{\mathtt{c}}[x \leftarrow [\![u]\!]^{\lambda}_{\mathtt{c}}]$ reduces to $(\overline{\alpha_i}\phi.[\![p]\!]^{\lambda}_{\mathtt{c}})[x \leftarrow [\![u]\!]^{\lambda}_{\mathtt{c}}]\ (\alpha_i\phi.[\![q]\!]^{\lambda}_{\mathtt{c}})$ by `apm`, where $\phi = \mathrm{fv}(q)$, and if $x \in t$ we can copy $[x \leftarrow [\![q]\!]^{\lambda}_{\mathtt{c}}]$ with `dup` and use an induction on ϕ to push a copy to $[\![p]\!]^{\lambda}_{\mathtt{c}}$ with `snd` — and conclude by induction. Finally, we can use $\equiv_g$ and $\equiv_s$ to apply `com` and go on by induction on q. □

There is however a problem with the translation, in the case where a reduction erases free variables, as some communication operators are made useless and thus are not present in the translation of the reduced term. In order to deal with this, we consider $\simeq$ the smallest congruence such that $(\overline{\alpha}x.t)\ (\alpha x.u) \simeq t\ u$ if $x \notin u$.

Theorem 4.4. *For any λ-terms t and u, if* $t \longrightarrow_{\beta} u$ *then* $[\![t]\!]^{\lambda}_{\mathtt{c}} \longrightarrow^{\equiv *}_{\lambda\mathtt{c}} v \simeq [\![u]\!]^{\lambda}_{\mathtt{c}}$.

Proof. If the β-reduction in t does not appear at toplevel, we use a straightforward induction on its structure, and in the case where t is of the shape $p\ q$, its translation reduces to some v where potentially more inputs and outputs are used than in $[\![u]\!]^{\lambda}_{\mathtt{c}}$ — because reduction might erase free variables — but such that $v \simeq [\![u]\!]^{\lambda}_{\mathtt{c}}$. Then, in the case where t is the redex $(\lambda x.p)\ q$ being reduced, we have:

$$\begin{array}{rcl} t \equiv (\lambda x.\overline{\alpha_i}\phi.[\![p]\!]^{\lambda}_{\mathtt{c}})\,(\alpha_i\phi.[\![q]\!]^{\lambda}_{\mathtt{c}}) & \longrightarrow_{\mathtt{B}} & (\overline{\alpha_i}\phi.[\![p]\!]^{\lambda}_{\mathtt{c}})[x \leftarrow \alpha_i\phi.[\![q]\!]^{\lambda}_{\mathtt{c}}] \\ & \longrightarrow^{*}_{\mathtt{opn}} & [\![p]\!]^{\lambda}_{\mathtt{c}}[x \leftarrow [\![q]\!]^{\lambda}_{\mathtt{c}}] \end{array}$$

so that we have finally $t \longrightarrow^{\equiv *}_{\lambda\mathtt{c}} [\![p\{q/x\}]\!]^{\lambda}_{\mathtt{c}}$, by Lemma 4.3. □

Projecting the dynamics of λc into β-reduction would be even more complex, because of *» backwards references «* allowed by communication — and preservation of strong normalisation is also made highly difficult in this setting.

Confluence. Due to the complex reduction behaviour of λc, confluence in this setting is bound to be a difficult result, valid only for some particular subset of all terms, better behaved than the others — as in λr, deadlocks disallowing a general confluence result. Notice however that the first and foremost problem here is the potentially non-deterministic use of channel names, leading to critical pairs which can clearly not be closed, as shown in the following term, in which two independent explicit substitutions are competing for a unique resource:

$$(\overline{\alpha}x.(y\ z))[x \leftarrow u][y \leftarrow \alpha w.w][z \leftarrow \alpha w.w]$$

so that specific conditions must be imposed on channels to obtain confluence through the $\longrightarrow^{\equiv}_{\lambda\mathtt{c}}$ relation, and possibly also conditions on the reducibility of terms.

$$\mathtt{var}\,\frac{}{x:\{\emptyset\}\triangleright A\vdash x:A}\qquad \mathtt{sub}\,\frac{\Gamma,x:\{\Delta\vdash u:A\}\triangleright A\vdash t:B}{\Gamma,\Delta\vdash t[x\leftarrow u]:B}$$

$$\mathtt{lam}\,\frac{\Gamma,x:A\vdash t:B}{\Gamma\vdash\lambda x.t:A\to B}\qquad \mathtt{ape}\,\frac{\Gamma\vdash t:(\{\vdash u:A\}\triangleright A)\to B}{\Gamma\vdash t\,u:B}$$

$$\mathtt{rem}\,\frac{\Gamma\vdash t:B}{\Gamma,x:A\vdash t:B}\qquad \mathtt{dup}\,\frac{\Gamma,x:\{\mathscr{U}\}\triangleright A,y:\{\mathscr{U}\}\triangleright A\vdash t_{[y/x]}:B}{\Gamma,x:\{\mathscr{U}\}\triangleright A\vdash t:B}$$

$$\mathtt{com}\,\frac{\Gamma\vdash t:(\{\Delta,y:A\vdash u:B\}\triangleright C)\to D}{\Gamma,x:A\vdash\overline{\alpha}x.t:(\{\Delta\vdash\alpha y.u:B\}\triangleright C)\to D}$$

$$\mathtt{cop}\,\frac{\Gamma,z:\{\Delta,y:A\vdash u:B\}\triangleright C\vdash t:D}{\Gamma,x:A,z:\{\Delta\vdash\alpha y.u:B\}\triangleright C\vdash\overline{\alpha}x.t:D}$$

Figure 4: Nested type system Nc for the λc-calculus

4.2 Nested Typing for λc

The type system Nc for the λc-calculus is the representation in terms of typing of the JD proof system for intuitionistic logic, extended with the cut rule used during detour elimination, as presented in Chapter 4. The explicit use of the switch rules in this system leads to the definition of typing rules for the pair of communication operators introduced in λc. Following the nested typing methodology, we will see how typing a term in this setting allows to ensure that it can be reduced. The basic notions are defined as in other nested type systems:

$$\mathscr{U}\ ::=\ x:\phi\vdash\mathscr{U}\ \mid\ t:T\ \mid\ \emptyset\qquad T\ ::=\ A\ \mid\ \phi\to T\qquad \phi\ ::=\ \{\mathscr{U}\}\triangleright A$$

We denote *plain types* A and *conditional types* T the same way, simply as formulas, for the sake of simplicity. The typing rules for the Nc system for λc are shown in Figure 4, and they are mostly the same as the Ns system for λs shown in Chapter 5, except for the equivalent of the switch rules, used to type communication operators.

The new typing rules used in Nc are rather different from rules of standard type systems, or other nested systems. Their intuitive meaning is the following:

- The com rule says that when typing a term $\overline{\alpha}x.t$ under an environment that contains the assumption $x:A$, if the type of this term depends on the typing of a term $\alpha y.u$, then the assumption $y:A$ must be used to type u, while the original $x:A$ must be erased before typing t.
- The cop rule is the same as the com rule except that it applies when the term $\alpha y.u$ is being typed in an assumption in the structure typing $\overline{\alpha}x.t$, rather than inside a conditional type.

From the viewpoint of the typing process, these rules com and cop are in charge of the dispatch of typing assumptions among all different terms being typed, within assumptions or inside conditional types. This explains the shape of the ape variant of the app rule, where no typing assumption is initially given to the structure in the premise, so that assumptions must be moved later inside, from the outer set of assumptions, to complete the typing process.

Example 4.5. *Below is shown an example derivation in the* Nc *type system, which is illustrating the use of both the* com *and* cop *typing rules, to move typing assumptions from outer parts of a typing judgement into the inner parts of this judgement, following a pair of communication operators.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y:A\to B\vdash y:A\to B}\,\texttt{var}\quad\emptyset}{y:\{\Delta\vdash u:A\to B\}\triangleright A\to B\vdash y:A\to B}\,\texttt{var}}{y:\{\Delta\vdash u:A\to B\}\triangleright A\to B\vdash y:(\{w:A\vdash w:A\}\triangleright A)\to B}\,\texttt{com}}{x:A,y:\{\Delta\vdash u:A\to B\}\triangleright A\to B\vdash \overline{\alpha}x.y:(\{\vdash \alpha w.w:A\}\triangleright A)\to B}\,\texttt{sub}}{\Delta,x:A\vdash(\overline{\alpha}x.y)[y\leftarrow u]:(\{\vdash \alpha w.w:A\}\triangleright A)\to B}\,\texttt{var}}{\Delta,x:\{n:A\vdash n:A\}\triangleright A\vdash(\overline{\alpha}x.y)[y\leftarrow u]:(\{\vdash \alpha w.w:A\}\triangleright A)\to B}\,\texttt{cop}}{\Delta,z:A,x:\{\vdash \kappa n.n:A\}\triangleright A\vdash\overline{\kappa}z.(\overline{\alpha}x.y)[y\leftarrow u]:(\{\vdash \alpha w.w:A\}\triangleright A)\to B}\,\texttt{app}}{\Delta,z:A,x:\{\vdash \kappa n.n:A\}\triangleright A\vdash(\overline{\kappa}z.(\overline{\alpha}x.y)[y\leftarrow u])\,(\alpha w.w):B}\,\texttt{sub}}{\Delta,z:A\vdash((\overline{\kappa}z.(\overline{\alpha}x.y)[y\leftarrow u])\,(\alpha w.w))[x\leftarrow\kappa n.n]:B}$$

Notice that the structure of the communication operators forces the typing process to build a derivation where the assumption $z:A$ is moved to the appropriate judgement, concerning x, before this assumption on x is itself moved to type the argument $(\alpha w.w)$. The typing derivation where the assumption on z is first moved into the context of the argument and then moved inside the judgement for x is obtained by considering the variant of the initial term where, in the body of the function, $\overline{\alpha}x$ appears above $\overline{\kappa}z$.

Despite its specificities, the Nc type system has properties similar to other nested type systems, and in particular, the typing process is terminating, as can be shown by defining an appropriate measure on typing structures, following the same scheme as in Nx and Ns. There is still a mismatch between terms and derivations, but one derivation is associated to only one λc-term.

The correspondence between the λc-calculus and the JD ∪ {u} system lies in the matching between reduction rules in λc and rewriting steps used in the detour elimination procedure used in JD ∪ {u}. As in the systems described in Chapter 5, we list the reduction rules of $\longrightarrow_{\lambda c}$ and observe how all the rewritings on a term correspond to rewritings on its typing derivation, and most cases are the same as in Ns or Nr. Note that in this setting, the ape rule cannot receive assumptions in the judgement it creates, so that instances of the com rule will stack upon it when it is permuted upwards, and we this β-redex is turned into a cut, the com instances will be stacked as well, although they can be removed later. We list the new cases:

1. The reduction rule $(t\ v)[x \leftarrow u] \longrightarrow_{\mathtt{apm}} t[x \leftarrow u]\ v$ reflects the exchange of a cut with an implication elimination, so that the derivation:

$$\mathtt{ape}\ \frac{\xi\{\Gamma, x : \{\ \mathscr{U}\ \} \triangleright B \vdash t : (\{\vdash v : C\} \triangleright C) \to A\}}{\xi\{\Gamma, x : \{\ \mathscr{U}\ \} \triangleright B \vdash t\ v : A\}} \qquad \mathscr{D}\Vert \qquad \mathtt{sub}\ \frac{\xi\{\Gamma, x : \{\Delta \vdash v : B\} \triangleright B \vdash t\ v : A\}}{\xi\{\Gamma, \Delta \vdash (t\ v)[x \leftarrow u] : A\}}$$

is turned in the following derivation:

$$\mathtt{sub}\ \frac{\mathtt{ape}\ \dfrac{\xi\{\Gamma, x : \{\ \mathscr{U}\ \} \triangleright B \vdash t : (\{\vdash v : C\} \triangleright C) \to A\}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u] : (\{\vdash v : C\} \triangleright C) \to A\}}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u]\ v : A\}} \qquad \mathscr{D}\Vert \qquad \frac{\xi\{\Gamma, x : \{\Delta \vdash u : B\} \triangleright B \vdash t : (\{\vdash v : C\} \triangleright C) \to A\}}{\xi\{\Gamma, \Delta \vdash t[x \leftarrow u] : (\{\vdash v : C\} \triangleright C) \to A\}}$$

2. The reduction rule $(\alpha y.t)[x \leftarrow u] \longrightarrow_{\mathtt{get}} \alpha y.t[x \leftarrow u]$ corresponds to the cut permuted above a switch moving some structure — not the one introduced by the cut — inside a conditional type or inside another structure, so that in the first case, the following derivation:

$$\mathtt{com}\ \frac{\xi\{\Gamma \vdash t : (\{\Psi, x : \{\ \mathscr{U}\ \} \triangleright A, z : B \vdash v : E\} \triangleright D) \to C\}}{\xi\{\Gamma, y : B \vdash \overline{\alpha} y.t : (\{\Psi, x : \{\ \mathscr{U}\ \} \triangleright A \vdash \alpha z.v : E\} \triangleright D) \to C\}} \qquad \mathscr{D}\Vert \qquad \mathtt{sub}\ \frac{\xi\{\Gamma, y : B \vdash \overline{\alpha} y.t : (\{\Psi, x : \{\Delta \vdash u : A\} \triangleright A \vdash \alpha z.v : E\} \triangleright D) \to C\}}{\xi\{\Gamma, y : B \vdash (\overline{\alpha} y.t) : (\{\Psi, \Delta \vdash (\alpha z.v)[x \leftarrow u] : E\} \triangleright D) \to C\}}$$

is turned into the derivation:

$$\mathtt{sub}\ \frac{\mathtt{com}\ \dfrac{\xi\{\Gamma \vdash t : (\{\Psi, x : \{\ \mathscr{U}\ \} \triangleright A, z : B \vdash v : E\} \triangleright D) \to C\}}{\xi\{\Gamma, y : B \vdash \overline{\alpha} y.t : (\{\Psi, x : \{\ \mathscr{U}\ \} \triangleright A \vdash \alpha z.v : E\} \triangleright D) \to C\}}}{\xi\{\Gamma, y : B \vdash \overline{\alpha} y.t : (\{\Psi, \Delta \vdash \alpha z.v[x \leftarrow u] : E\} \triangleright D) \to C\}} \qquad \mathscr{D}\Vert \qquad \frac{\xi\{\Gamma \vdash t : (\{\Psi, x : \{\Delta \vdash u : A\} \triangleright A, z : B \vdash v : E\} \triangleright D) \to C\}}{\xi\{\Gamma \vdash t : (\{\Psi, \Delta, z : B \vdash v[x \leftarrow u] : E\} \triangleright D) \to C\}}$$

and in the second case, the derivation:

$$\mathtt{cop}\ \frac{\xi\{\Gamma, w : \{\Psi, x : \{\ \mathscr{U}\ \} \triangleright A, z : B \vdash v : E\} \triangleright D \vdash t : C\}}{\xi\{\Gamma, y : B, w : \{\Psi, x : \{\ \mathscr{U}\ \} \triangleright A \vdash \alpha z.v : E\} \triangleright D \vdash \overline{\alpha} y.t : C\}} \qquad \mathscr{D}\Vert \qquad \mathtt{sub}\ \frac{\xi\{\Gamma, y : B, w : \{\Psi, x : \{\Delta \vdash u : A\} \triangleright A \vdash \alpha z.v : E\} \triangleright D \vdash \overline{\alpha} y.t : C\}}{\xi\{\Gamma, y : B, w : \{\Psi, \Delta \vdash (\alpha z.v)[x \leftarrow u] : E\} \triangleright D \vdash (\overline{\alpha} y.t) : C\}}$$

is turned into the derivation:

$$\dfrac{\dfrac{\xi\{\Gamma, w:\{\Psi, x:\{\mathscr{U}\}\triangleright A, z:B\vdash v:E\}\triangleright D\vdash t:C\}}{\mathscr{D}\Vert}}{\dfrac{\xi\{\Gamma, w:\{\Psi, x:\{\Delta\vdash u:A\}\triangleright A, z:B\vdash v:E\}\triangleright D\vdash t:C\}}{\dfrac{\xi\{\Gamma, w:\{\Psi,\Delta, z:B\vdash v[x\leftarrow u]:E\}\triangleright D\vdash t:C\}}{\xi\{\Gamma, y:B, w:\{\Psi,\Delta\vdash \alpha z.v[x\leftarrow u]:E\}\triangleright D\vdash \overline{\alpha}y.t:C\}}\,\mathtt{cop}}\,\mathtt{sub}}$$

3. The reduction rule $(\overline{\alpha}y.t)[x\leftarrow u]\longrightarrow_{\mathtt{snd}}\overline{\alpha}y.t[x\leftarrow u]$ corresponds to the the other configuration of a cut permuted above an unrelated switch moving a structure inside inside a conditional type or inside another structure, so that in the first case, the following derivation:

$$\dfrac{\dfrac{\xi\{\Gamma, x:\{\mathscr{U}\}\triangleright A\vdash t:(\{\Psi, z:B\vdash v:E\}\triangleright D)\to C\}}{\xi\{\Gamma, y:B, x:\{\mathscr{U}\}\triangleright A\vdash \overline{\alpha}y.t:(\{\Psi\vdash \alpha z.v:E\}\triangleright D)\to C\}}\,\mathtt{com}}{\dfrac{\xi\{\Gamma, y:B, x:\{\Delta\vdash u:A\}\triangleright A\vdash \overline{\alpha}y.t:(\{\Psi\vdash \alpha z.v:E\}\triangleright D)\to C\}}{\xi\{\Gamma, y:B,\Delta\vdash(\overline{\alpha}y.t)[x\leftarrow u]:(\{\Psi\vdash \alpha z.v:E\}\triangleright D)\to C\}}\,\mathtt{sub}}$$

is turned into the derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma, x:\{\mathscr{U}\}\triangleright A\vdash t:(\{\Psi, z:B\vdash v:E\}\triangleright D)\to C\}}{\mathscr{D}\Vert}}{\dfrac{\xi\{\Gamma, x:\{\Delta\vdash u:A\}\triangleright A\vdash t:(\{\Psi, z:B\vdash v:E\}\triangleright D)\to C\}}{\xi\{\Gamma,\Delta\vdash t[x\leftarrow u]:(\{\Psi, z:B\vdash v:E\}\triangleright D)\to C\}}\,\mathtt{sub}}}{\xi\{\Gamma, y:B,\Delta\vdash\overline{\alpha}y.t[x\leftarrow u]:(\{\Psi\vdash\alpha z.v:E\}\triangleright D)\to C\}}\,\mathtt{com}$$

and in the second case, the derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma, x:\{\mathscr{U}\}\triangleright A, w:\{\Psi, z:B\vdash v:E\}\triangleright D\vdash t:C\}}{\xi\{\Gamma, y:B, x:\{\mathscr{U}\}\triangleright A, w:\{\Psi\vdash\alpha z.v:E\}\triangleright D\vdash\overline{\alpha}y.t:C\}}\,\mathtt{cop}}{\mathscr{D}\Vert}}{\dfrac{\xi\{\Gamma, y:B, x:\{\Delta\vdash u:A\}\triangleright A, w:\{\Psi\vdash\alpha z.v:E\}\triangleright D\vdash\overline{\alpha}y.t:C\}}{\xi\{\Gamma, y:B,\Delta, w:\{\Psi\vdash\alpha z.v:E\}\triangleright D\vdash(\overline{\alpha}y.t)[x\leftarrow u]:C\}}\,\mathtt{sub}}$$

is turned into the derivation:

$$\dfrac{\dfrac{\dfrac{\xi\{\Gamma, x:\{\mathscr{U}\}\triangleright A, w:\{\Psi, z:B\vdash v:E\}\triangleright D\vdash t:C\}}{\mathscr{D}\Vert}}{\dfrac{\xi\{\Gamma, x:\{\Delta\vdash u:A\}\triangleright A,\{\Psi, z:B\vdash v:E\}\triangleright D\vdash t:C\}}{\xi\{\Gamma,\Delta, w:\{\Psi, z:B\vdash v:E\}\triangleright D\vdash t[x\leftarrow u]:C\}}\,\mathtt{sub}}}{\xi\{\Gamma, y:B,\Delta, w:\{\Psi\vdash\alpha z.v:E\}\triangleright D\vdash\overline{\alpha}y.t[x\leftarrow u]:C\}}\,\mathtt{cop}$$

Finally the principal cases involving the switch rules correspond to the case of the communication rules, and in each case this just a linear permutation of a cut above a switch moving this cut.

4. The reduction $\pi\{(\overline{\alpha}x.t)[x \leftarrow u]\}\,(\alpha y.v) \longrightarrow_{\mathtt{com}} \pi\{\overline{\alpha_i}\phi.t\}\,(\alpha_i\phi.v[x \leftarrow u])$ is the reflection of the exchange of a cut with a switch, so that the derivation:

$$\mathtt{com}\,\frac{\xi\{\Gamma \vdash t : (\{\,\Psi, y : \{\,\mathscr{U}\,\} \triangleright A \vdash v : B\,\} \triangleright C) \to D\}}{\xi\{\Gamma, x : \{\,\mathscr{U}\,\} \triangleright A \vdash \overline{\alpha}x.t : (\{\,\Psi \vdash \alpha y.v : B\,\} \triangleright C) \to D\}}$$
$$\mathscr{D}\Big\|$$
$$\mathtt{sub}\,\frac{\xi\{\Gamma, x : \{\,\Delta \vdash u : A\,\} \triangleright A \vdash \overline{\alpha}x.t : (\{\,\Psi \vdash \alpha y.v : B\,\} \triangleright C) \to D\}}{\xi\{\Gamma, \Delta \vdash (\overline{\alpha}x.t)[x \leftarrow u] : (\{\,\Psi \vdash \alpha y.v : B\,\} \triangleright C) \to D\}}$$

is turned in the following derivation:

$$\xi\{\Gamma \vdash t : (\{\,\Psi, y : \{\,\mathscr{U}\,\} \triangleright A \vdash v : B\,\} \triangleright C) \to D\}$$
$$\mathscr{D}\Big\|$$
$$\mathtt{sub}\,\frac{\xi\{\Gamma \vdash t : (\{\,\Psi, x : \{\,\Delta \vdash u : A\,\} \triangleright A \vdash v : B\,\} \triangleright C) \to D\}}{\xi\{\Gamma \vdash t : (\{\,\Psi, \Delta \vdash v[x \leftarrow u] : B\,\} \triangleright C) \to D\}}$$
$$\mathtt{com}^*\,\frac{\xi\{\Gamma \vdash t : (\{\,\Psi, \Delta \vdash v[x \leftarrow u] : B\,\} \triangleright C) \to D\}}{\xi\{\Gamma, \Delta \vdash \overline{\alpha_i}\phi.t : (\{\,\Psi \vdash \alpha_i\phi.v[x \leftarrow u] : B\,\} \triangleright C) \to D\}}$$

5. The rule $\vec{\pi}\{(\overline{\alpha}x.t)[x \leftarrow u]\}[z \leftarrow \alpha y.v] \longrightarrow_{\mathtt{cop}} \vec{\pi}\{\overline{\alpha_i}\phi.t\}[z \leftarrow \alpha_i\phi.v[x \leftarrow u]]$ is the reflection of the exchange of a cut with an instance of the other form of switch, so that the derivation:

$$\mathtt{cop}\,\frac{\xi\{\Gamma, z : \{\,\Psi, y : \{\,\mathscr{U}\,\} \triangleright A \vdash v : B\,\} \triangleright C \vdash t : D\}}{\xi\{\Gamma, x : \{\,\mathscr{U}\,\} \triangleright A, z : \{\,\Psi \vdash \alpha y.v : B\,\} \triangleright C \vdash \overline{\alpha}x.t : D\}}$$
$$\mathscr{D}\Big\|$$
$$\mathtt{sub}\,\frac{\xi\{\Gamma, x : \{\,\Delta \vdash u : A\,\} \triangleright A, z : \{\,\Psi \vdash \alpha y.v : B\,\} \triangleright C \vdash \overline{\alpha}x.t : D\}}{\xi\{\Gamma, \Delta, z : \{\,\Psi \vdash \alpha y.v : B\,\} \triangleright C \vdash (\overline{\alpha}x.t)[x \leftarrow u] : D\}}$$

is turned in the following derivation:

$$\xi\{\Gamma, z : \{\,\Psi, y : \{\,\mathscr{U}\,\} \triangleright A \vdash v : B\,\} \triangleright C \vdash t : D\}$$
$$\mathscr{D}\Big\|$$
$$\mathtt{sub}\,\frac{\xi\{\Gamma, z : \{\,\Psi, x : \{\,\Delta \vdash u : A\,\} \triangleright A \vdash v : B\,\} \triangleright C \vdash t : D\}}{\xi\{\Gamma, z : \{\,\Psi, \Delta \vdash v[x \leftarrow u] : B\,\} \triangleright C \vdash t : D\}}$$
$$\mathtt{cop}^*\,\frac{\xi\{\Gamma, z : \{\,\Psi, \Delta \vdash v[x \leftarrow u] : B\,\} \triangleright C \vdash t : D\}}{\xi\{\Gamma, \Delta, z : \{\,\Psi \vdash \alpha_i\phi.v[x \leftarrow u] : B\,\} \triangleright C \vdash \overline{\alpha_i}\phi.t : D\}}$$

6. The rule $(\overline{\alpha}z.t)[x \leftarrow \alpha y.u] \longrightarrow_{\mathtt{opn}} t[x \leftarrow u\{z/y\}]$ reflects the assimilation of a switch instance into a cut, when this switch moves another structure into the structure introduced by the cut, so that the derivation:

$$\mathtt{cop}\,\frac{\xi\{\Gamma, x : \{\,\Psi, y : \{\,\mathscr{U}\,\} \triangleright B \vdash u : A\,\} \triangleright A \vdash t : C\}}{\xi\{\Gamma, z : \{\,\mathscr{U}\,\} \triangleright B, x : \{\,\Psi \vdash \alpha y.u : A\,\} \triangleright A \vdash \overline{\alpha}z.t : C\}}$$
$$\mathscr{D}\Big\|$$
$$\mathtt{sub}\,\frac{\xi\{\Gamma, z : \{\,\mathscr{U}\,\} \triangleright B, x : \{\,\Delta \vdash \alpha y.u : A\,\} \triangleright A \vdash \overline{\alpha}z.t : C\}}{\xi\{\Gamma, z : \{\,\mathscr{U}\,\} \triangleright B, \Delta \vdash (\overline{\alpha}z.t)[x \leftarrow \alpha y.u] : C\}}$$

is turned in the following derivation:

$$\mathtt{sub}\,\dfrac{\xi\{\Gamma, x:\{\Psi, z:\{\mathscr{U}\}\triangleright B\vdash u\{z/y\}:A\}\triangleright A\vdash t:C\}}{\xi\{\Gamma, x:\{\Delta, z:\{\mathscr{U}\}\triangleright B\vdash u\{z/y\}:A\}\triangleright A\vdash t:C\}}\,\mathscr{D}\Vert$$
$$\xi\{\Gamma, z:\{\mathscr{U}\}\triangleright B, \Delta\vdash t[x\leftarrow u\{z/y\}]:C\}$$

In the λc-calculus, the composition of explicit substitution is performed through the `cop` rule, which was described as a non-trivial permutation, or with the `cmp` rule, which was a trivial permutation. The exchange of two substitutions, obtained by the equation $\equiv_e$ corresponds to another trivial permutation, where unrelated cuts are moved one above the other. Finally, the equations $\equiv_g$ and $\equiv_s$ correspond to trivial permutations between two instances of the same kind of switch rule, and $\equiv_i$ and $\equiv_r$ to the trivial permutation of an implication introduction with both kinds of switches. The correspondence between reduction in λc and detour elimination in $\mathsf{JD}\cup\{\mathtt{u}\}$ described in this case analysis and in Chapter 5 leads us to the conclusion that if a λc-term admits a typing derivation, there exists a strategy for reducing it into a normal form.

Theorem 4.6 (Subject reduction)**.** *If* $\Gamma\vdash t:A$ *and* $t\longrightarrow_{\lambda\mathtt{c}} u$ *then* $\Gamma\vdash u:A$*.*

Proof. This result can be obtained by inspection of the rewriting cases shown above, since each reduction rule in the λc-calculus corresponds to a case of moving a cut above another rule instance — except the `B`, rule which is the transformation of an introduction and elimination pair in a cut. □

Theorem 4.7 (Normalisation)**.** *For any λc-term t, if* $\Gamma\vdash t:A$ *there is a λc-term u such that we have* $t\longrightarrow^*_{\lambda\mathtt{c}} u$ *and u is a normal form.*

Proof. This immediately follows from the termination of the procedure for detour elimination in Chapter 4 and Theorem 4.6, since any reduction step corresponds to a detour elimination step applied in the typing derivation. Moreover, by definition, when a typing derivation is normal, we know that the corresponding term contains no explicit substitution and no β-redex. □

This result is rather weak, due to the treatment of channel names in λc and in the type system $\mathtt{Nc}$, and in general because of the lack of good properties of the whole set of untyped λc-terms. Studying further the kind of guarantees offered by typing might offer a more satisfying situation, for example if typing can be proved to ensure reducibility — despite non-confluence, which would in addition require conditions on the use of channel names.

PART 4

Nested Proof Search as Computation

Chapter 7

Nested Focusing in Linear Logic

In this chapter, we consider the transfer of the *focusing* technique from the LL sequent calculus for linear logic where it has its roots, to the setting of the calculus of structures, with the double purpose of proving that focusing is not essentially a feature of the sequent calculus formalism, while exploring the possibility of using a nested deduction system to perform structured proof search. Indeed, there are two sides to the focusing concept: it is a normal form — and the calculus of structures offers a versatile notion of proof supporting many interesting normal forms — but also a technique for efficient proof search — on this point, the calculus of structures is *a priori* problematic, because of the huge search space it induces.

In order to define a notion of focusing that would follow the principles of nested deduction, we need to be able to observe the same decompositions in a focused calculus of structures as in an unfocused one, and in particular we need to preserve the switch decomposition. The approach used here is to transfer in the calculus of structures the essence of the focusing notion: the respect of the *polarities*. We show here how the introduction of explicit polarity shifts in a system for linear logic leads to the observation that not all rules respect the polarity of their conclusion. Then, a modification of these rules induces a system having a property similar to the subformula property, but ensuring that no rule introduces new polarity shifts from conclusion to premise — this can be used as a characterisation of the notion of focused proof in the calculus of structures, by analogy with analytic proofs.

Since our goal is also to make proof search reasonable in the nested deduction setting, we start with the usual system for linear logic in the calculus of structures and remove all equations, introducing the necessary rules to perform the deduction steps that were previously implicit. Then, we refine this system in several steps to incorporate formulas with explicit polarities and use only inference rules respecting the polarities in its conclusive formula. Finally, we describe a focused calculus of structures, as well as a grouped system using synthetic positive rules. The grouped system is then used to provide a surprisingly simple proof of completeness of the focusing normal form, and we also discuss the relation of the focused calculus of structures to the standard focused sequent calculus for linear logic.

1 Linear Logic in the Calculus of Structures

As it was mentioned in Chapter 2, the standard proof system for linear logic [Gir87] is given in the form of a sequent calculus, that can be derived from any variant of the sequent calculus LK by a careful analysis of structural rules and polarities. The definition of a proof system implementing full linear logic under the deep inference methodology was done in the calculus of structures [Str03a], the system being called SLS, and this is our starting point.

There are several ways of defining the inference rules for linear logic, leading to variants of the SLS system. Here, we have particular needs to refine the system into its focused form, so that we will choose the shape of the rules accordingly. To keep notations simple, we overload the name SLS to denote our own variant[1].

1.1 The Symmetric Linear System SLS

The level of formulas for linear logic in the calculus of structures is exactly the same as in the sequent calculus. We assume given a countable set of atoms, denoted by letters such as a, b and c. Then, we need another set of *negated* atoms, isomorphic to the first one, and we denote by $\overline{a}$, $\overline{b}$ and $\overline{c}$ the negated atoms corresponding to atoms a, b and c. We also have the usual linear disjunctions $\parr$ and $\oplus$, the linear conjunctions $\otimes$ and $\with$, and the corresponding units $\bot$ and 0, and 1 and $\top$. Finally, we have the exponential modalities ! and ? to control infinite behaviour.

Definition 1.1. *The* formulas *of linear logic are defined by the following grammar:*

$$A,B ::= a \mid \overline{a} \mid \bot \mid 1 \mid A \parr B \mid A \otimes B \mid \top \mid 0 \mid A \with B \mid A \oplus B \mid ?A \mid !A$$

and structures *of linear logic are defined as the equivalence classes of formulas through the congruence induced by the equations shown in Figure 1.*

The role of the congruence relation, denoted by $\equiv$, is to keep notations simple and proofs readable, since we can avoid defining the corresponding inference rules — this is similar to the use of mutisets in the sequent calculus. Moreover, negation in linear logic is involutive, and we can push it to the atoms, where it is expressed through the two isomorphic sets of atoms described above, by De Morgan's laws of dualities between connectives.

Definition 1.2. Linear negation *is defined on structures by the following equations:*

$$\begin{array}{llllll} A^{\bot\bot} = A & a^\bot = \overline{a} & \bot^\bot = 1 & (A \parr B)^\bot = A^\bot \otimes B^\bot \\ & (?A)^\bot = !A^\bot & 0^\bot = \top & (A \oplus B)^\bot = A^\bot \with B^\bot \end{array}$$

Although dualities are important, the use of the linear negation operator is not often needed, as only atoms must be marked as negated. Notice that the equations shown above define negation for all connectives and units, because of involutivity of the negation. For example, we have $1^\bot = \bot^{\bot\bot}$ and $\bot^{\bot\bot} = \bot$, therefore $1^\bot = \bot$, and $(A^\bot \otimes B^\bot)^\bot = (A \parr B)^{\bot\bot}$, so that $(A \otimes B)^\bot = A^\bot \parr B^\bot$.

[1] This variant differs on the design of some inference rules, not essentially in the general design, but we will prove soundness and completeness from the sequent calculus, for the sake of clarity.

$$
\begin{array}{rcl@{\qquad}rcl}
A \parr B & \equiv & B \parr A & A \otimes B & \equiv & B \otimes A \\
A \with B & \equiv & B \with A & A \oplus B & \equiv & B \oplus A \\
A \parr (B \parr C) & \equiv & (A \parr B) \parr C & A \otimes (B \otimes C) & \equiv & (A \otimes B) \otimes C \\
A \with (B \with C) & \equiv & (A \with B) \with C & A \oplus (B \oplus C) & \equiv & (A \oplus B) \oplus C \\
\bot \parr A & \equiv & A & 1 \otimes A & \equiv & A \\
\top \with A & \equiv & A & 0 \oplus A & \equiv & A \\
?\bot & \equiv & \bot & !1 & \equiv & 1 \\
1 \with 1 & \equiv & 1 & \bot \oplus \bot & \equiv & \bot
\end{array}
$$

Figure 1: Equations for linear structures

In order to have the ability to apply inference rules deep inside structures, we define contexts as usual, by indicating where the hole is within a structure. There is no restriction here where this hole can be, unlike the intuitionistic system presented in Chapter 4, since we are working with *classical* linear logic.

Definition 1.3. *The* contexts *of linear logic are structures with a hole* $\{\ \}$*, meant to be filled by another structure and defined by the following grammar:*

$$\xi ::= \{\ \} \mid \xi \parr A \mid \xi \otimes A \mid \xi \with A \mid \xi \oplus A \mid ?\xi \mid !\xi$$

The inference rules for SLS are given in Figure 2, where no context is explicity written, but all of them can be applied deep inside a structure. Both the basic *down fragment* LS and the *up fragment* are shown, with the *switch* rule s being self-dual, so that it belongs to both fragments. As usual, the most important rule from the up fragment is $\mathsf{i}\uparrow$, the equivalent of the cut rule in the sequent calculus. It can be used to encode other up rules, using their dual down rules. Notice that a proof of some structure A in this system is a derivation from 1 to A — the additive truth unit $\top$ is not considered a valid final premise, but it can be deduced from 1 using the $\mathsf{u}\downarrow$ rule. The inference rules for this system are almost the same as the ones of the original system [Str03a], with some minor differences:

- we use explicity the inference rule $\mathsf{a}\downarrow$ to handle erasures induced by $\top$, since we intend not to use the identity rule $\mathsf{i}\downarrow$ on units — this also forces us to use the $\mathsf{u}\downarrow$ rule to remove additive units.
- additive contraction is built inside the $\mathsf{y}\downarrow$ rule for $\with$, rather than attached to the $\oplus$ connective so that the interaction between additive connectives is not a primitive operation:

$$\frac{\xi\{(A \parr C) \with (B \parr D)\}}{\xi\{(A \oplus B) \parr (C \with D)\}} \quad = \quad \mathsf{o}\downarrow \frac{\mathsf{o}\downarrow \dfrac{\xi\{(A \parr C) \with (B \parr D)\}}{\xi\{(A \parr C) \with ((A \oplus B) \parr D)\}}}{\mathsf{y}\downarrow \dfrac{\xi\{((A \oplus B) \parr C) \with ((A \oplus B) \parr D)\}}{\xi\{(A \oplus B) \parr (C \with D)\}}}$$

Multiplicatives

$$\mathsf{i}{\downarrow}\,\frac{}{A^\perp \parr A} \qquad \mathsf{s}\,\frac{(A \parr B) \otimes C}{A \parr (B \otimes C)} \qquad \mathsf{i}{\uparrow}\,\frac{A^\perp \otimes A}{\bot}$$

Additives

$$\mathsf{a}{\downarrow}\,\frac{\top}{\top \parr A} \qquad \mathsf{o}{\downarrow}\,\frac{A}{A \oplus B} \qquad\Big|\qquad \mathsf{o}{\uparrow}\,\frac{A \mathbin{\&} B}{A} \qquad \mathsf{a}{\uparrow}\,\frac{0 \otimes A}{0}$$

$$\mathsf{y}{\downarrow}\,\frac{(A \parr C) \mathbin{\&} (B \parr C)}{(A \mathbin{\&} B) \parr C} \qquad \mathsf{u}{\downarrow}\,\frac{1}{\top} \qquad\Big|\qquad \mathsf{u}{\uparrow}\,\frac{0}{\bot} \qquad \mathsf{y}{\uparrow}\,\frac{(A \oplus C) \otimes B}{(A \otimes B) \oplus (C \otimes B)}$$

Exponentials

$$\mathsf{c}{\downarrow}\,\frac{?A \parr ?A}{?A} \qquad \mathsf{w}{\downarrow}\,\frac{\bot}{?A} \qquad\Big|\qquad \mathsf{w}{\uparrow}\,\frac{!A}{1} \qquad \mathsf{c}{\uparrow}\,\frac{!A}{!A \otimes !A}$$

$$\mathsf{p}{\downarrow}\,\frac{!(?A \parr B)}{?A \parr !B} \qquad \mathsf{d}{\downarrow}\,\frac{A}{?A} \qquad\Big|\qquad \mathsf{d}{\uparrow}\,\frac{!A}{A} \qquad \mathsf{p}{\uparrow}\,\frac{!A \otimes ?B}{?(!A \otimes B)}$$

Figure 2: Inference rules for system SLS

- the thinning rule t↓ is replaced with the more standard rule o↓ to handle the $\oplus$ connective, but this does not affect the shape of the proofs, since the 0 unit can only be removed by the equation $0 \oplus A \equiv A$.
- exponential contraction is separated from dereliction, so that c has two copies of $?A$ as a premise and we need d↓ to obtain a copy of A without a modality:

$$\frac{\xi\{?A \parr A\}}{\xi\{?A\}} \quad = \quad \begin{array}{l}\mathsf{d}{\downarrow}\,\dfrac{\xi\{?A \parr A\}}{\xi\{?A \parr ?A\}} \\ \mathsf{c}{\downarrow}\,\dfrac{\xi\{?A \parr ?A\}}{\xi\{?A\}}\end{array}$$

- we also separate dereliction from the promotion rule, so that the ? modality appears in the premise of the p↓ rule, and we can obtain the other form of promition using the d↓ rule:

$$\frac{\xi\{!(A \parr B)\}}{\xi\{?A \parr !B\}} \quad = \quad \begin{array}{l}\mathsf{d}{\downarrow}\,\dfrac{\xi\{!(A \parr B)\}}{\xi\{!(?A \parr B)\}} \\ \mathsf{p}{\downarrow}\,\dfrac{\xi\{!(?A \parr B)\}}{\xi\{?A \parr !B\}}\end{array}$$

- symmetrically, the differences on up rules are the same as the differences we have described for down rules.

Remark 1.4. *The inference rules of the* SLS *system are divided into three categories, multiplicatives, additives and exponentials, corresponding to the three* layers *of linear logic, but they are also divided into the down fragment, shown on the left, and the up fragment, shown on the right, where* s *in the middle belongs to both.*

This system is the basis for all other systems of linear logic we use here. It has the same properties as the original system, and in particular, completeness is not lost when restricting the identity i↓ and cut i↑ rules to their atomic form.

Proposition 1.5. *Any instance of the* i↓ *rule can be replaced by a derivation in* LS *with same premise and conclusion, using instances of* i↓ *only in its atomic form* ai↓.

Proof. By induction on the structure A affected by a general instance of the identity rule, with premise $\xi\{1\}$ and conclusion $\xi\{A^{\perp} \parr A\}$. If A is an atom, then we are done. Otherwise, we use a case analysis on the shape of A, and build a derivation using identity instances on smaller structures, and other rules depending on the toplevel connective of A — for example, for $\parr$ or $\otimes$ we use a switch s, for $\with$ or $\oplus$ we use the y↓ and o↓ rules. In the cases where A is some unit, we build a derivation directly with the rules a↓ and u↓, and the congruence. □

The set of equations provided to define the congruence $\equiv$ is not minimal, but it was chosen to avoid writing down obvious equivalence in inference steps. We will see later which equations are required to have a complete system.

Remark 1.6. *Using a congruence implicitly on formulas is not a problem here, because the equations we use correspond to linear equivalences. A sequence of applications of inference rules can also be made more explicit by using the fake rule* $\equiv$*, which can be instantiated with premise A and conclusion B whenever* $A \equiv B$*. Moreover, a structure always has a normal form [Str03a] — for example,* $A \parr ?\perp$ *can be written A.*

1.2 Correspondence to the Sequent Calculus

In order to get a better insight on the way the SLS system compares to the standard sequent calculus for linear logic, we will now show soundness and completeness of SLS with respect to the LL sequent calculus shown in Figure 3. This is done using a translation in each direction, between sequents and structures, and we can then build from any proof in one system a corresponding proof in the other system. For soundness, the translation from structures to sequent is trivial.

Theorem 1.7 (Soundness of SLS)**.** *If some structure A is provable in* SLS*, then the sequent* $\vdash A$ *is provable in the* $\mathsf{LL} \cup \{\mathsf{cut}\}$ *sequent calculus.*

Proof. By induction on the length of a given proof $\mathscr{P}$ of A in SLS. In the base case, the proof $\mathscr{P}$ is reduced to the structure 1, we use an instance of the 1 rule, and we are done. In the general case, we consider the bottommost instance r in $\mathscr{P}$:

$$\mathsf{r}\,\frac{\begin{array}{c}\vdots\\ \xi\{C\}\end{array}}{\xi\{B\}}$$

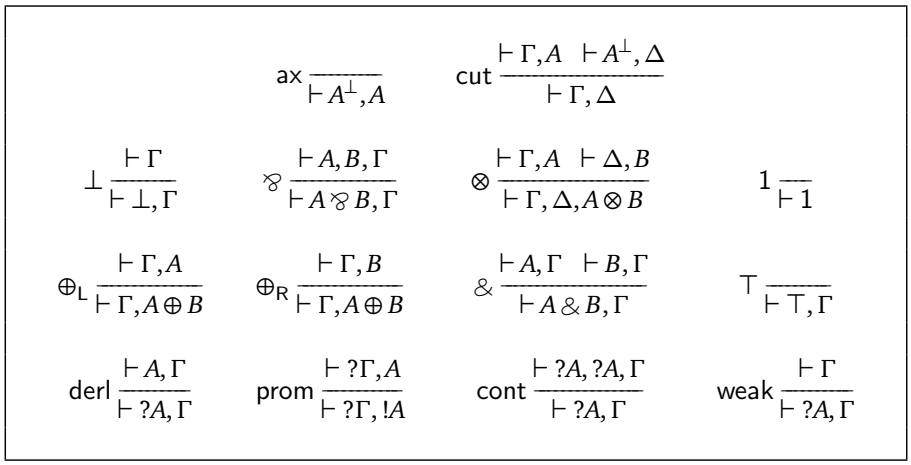


Figure 3: Inference rules for system $\mathsf{LL} \cup \{\mathsf{cut}\}$

and we must show that the linear implication $\xi\{C\} \multimap \xi\{B\}$ holds in the $\mathsf{LL} \cup \{\mathsf{cut}\}$ sequent calculus, when written as the equivalent formula $\xi\{C\}^\perp \parr \xi\{B\}$. The first step is to show that the sequent $\vdash C^\perp, B$ is provable in $\mathsf{LL} \cup \{\mathsf{cut}\}$, by using a case analysis on the r instance, and building the corresponding proof. In each case, we can easily build such a proof. Then, by straightforward induction on the context ξ, we show that this proof can be transformed into a proof Π_1 of $\vdash \xi\{C\}^\perp, \xi\{B\}$. We conclude by producing a proof Π_2 of $\vdash \xi\{C\}$ by induction hypothesis, and use the two proofs to build the expected proof of $\vdash \xi\{B\}$ in $\mathsf{LL} \cup \{\mathsf{cut}\}$:

$$\mathsf{cut}\ \frac{\overset{\Pi_2}{\vdash \xi\{C\}} \qquad \overset{\Pi_1}{\vdash \xi\{C\}^\perp, \xi\{B\}}}{\vdash \xi\{B\}}$$

□

Remark 1.8. *The translation from structures to sequents relies on the extraction of a formula from a structure, which is of course always possible. However, the formula used as translation is not unique, because of the many equations of the congruence. A more precise statement for soundness would thus be that any sequent $\vdash A$ obtained by translation is provable. Fortunately, this is obvious because equations are equivalences in linear logic, and we prove completeness of* SLS *below, so that if $A \equiv B$ and $\vdash A$ is provable, then $\vdash B$ is provable.*

The translation in the other direction, needed to prove completeness of SLS, is not direct as the one for soundness. But sequents in $\mathsf{LL} \cup \{\mathsf{cut}\}$ are simple objects, so that this boils down to the replacement of commas with $\parr$ connectives.

Definition 1.9. *The translation $[\![\cdot]\!]_\mathsf{P}$ from linear sequents into linear structures is defined recursively as follows:*

$$[\![\vdash A]\!]_\mathsf{P} = A \qquad \textit{and} \qquad [\![\vdash A, \Gamma]\!]_\mathsf{P} = A \parr [\![\Gamma]\!]_\mathsf{P}$$

We can now prove the completeness theorem, by translation from the sequent calculus. The proof is a good illustration of the inference mechanism in SLS, and in particular it shows how nesting replaces the branching abilities of sequent calculi.

Theorem 1.10 (Completeness of SLS)**.** *If a sequent* $\vdash \Gamma$ *is provable in the* $\mathsf{LL} \cup \{\mathsf{cut}\}$ *sequent calculus, then the structure* $[\![\Gamma]\!]_{\mathsf{P}}$ *is provable in* SLS.

Proof. By induction on a proof Π of the sequent $\vdash \Gamma$ in $\mathsf{LL} \cup \{\mathsf{cut}\}$, and case analysis on the bottommost rule instance r in Π, we build a proof $\mathscr{P}$ of the translation of this sequent in SLS. This follows a simple scheme, where the translation of branches of the sequent calculus are composed by the connective corresponding to the rule used, the congruence is used to handle most units, and antecedents are distributed using the switch and duplication rules. In the base case, Π is an axiomatic instance, and the result is immediate:

$$\mathsf{ax}\,\frac{}{\vdash A^{\perp}, A} \quad\longrightarrow\quad \mathsf{i}{\downarrow}\,\frac{1}{A^{\perp} \parr A} \qquad \text{and} \qquad \top\,\frac{}{\vdash \top, \Delta} \quad\longrightarrow\quad \mathsf{a}{\downarrow}\,\frac{\mathsf{u}{\downarrow}\,\dfrac{1}{\top}}{\top \parr [\![\Delta]\!]_{\mathsf{P}}}$$

and an instance of the 1 rule is simply translated into the structure 1, which in itself is a proof. In the general case, the induction hypothesis needs to be used, as shown in the following example:

$$\otimes\,\frac{\overset{\Pi_A}{\vdash \Delta, A} \qquad \overset{\Pi_B}{\vdash \Psi, B}}{\vdash \Delta, \Psi, A \otimes B} \quad\longrightarrow\quad \mathsf{s}\,\frac{\mathsf{s}\,\dfrac{1 \;\Vdash \mathscr{D}_A}{[\![\Delta]\!]_{\mathsf{P}} \parr A \equiv [\![\Delta]\!]_{\mathsf{P}} \parr (A \otimes 1)} \;\Vdash \mathscr{D}_B}{[\![\Delta]\!]_{\mathsf{P}} \parr (A \otimes ([\![\Psi]\!]_{\mathsf{P}} \parr B)) \equiv [\![\Delta]\!]_{\mathsf{P}} \parr [\![\Psi]\!]_{\mathsf{P}} \parr (A \otimes B)}$$

where $\mathscr{D}_A$ and $\mathscr{D}_B$ are obtained by induction hypothesis from the proofs Π_A and Π_B respectively, and composed by a $\otimes$ using the deep inference methodology, which allows to plug a proof in a context. The cases of other rules for multiplicative and additive connectives and units are similar, and rules for exponentials are directly translated, since none of them is branching. The identity rule ax is also immediately translated using the $\mathsf{i}{\downarrow}$ rule. Symmetrically, the cut rule cut is translated by the $\mathsf{i}{\uparrow}$ rule, and this case is almost the same as the $\otimes$ case:

$$\mathsf{cut}\,\frac{\overset{\Pi_1}{\vdash \Delta, A} \qquad \overset{\Pi_2}{\vdash A^{\perp}, \Psi}}{\vdash \Delta, \Psi} \quad\longrightarrow\quad \mathsf{i}{\uparrow}\,\frac{\mathsf{s}\,\dfrac{\mathsf{s}\,\dfrac{1 \;\Vdash \mathscr{D}_1}{[\![\Delta]\!]_{\mathsf{P}} \parr A \equiv [\![\Delta]\!]_{\mathsf{P}} \parr (1 \otimes A)} \;\Vdash \mathscr{D}_2}{[\![\Delta]\!]_{\mathsf{P}} \parr (([\![\Psi]\!]_{\mathsf{P}} \parr A^{\perp}) \otimes A)}}{[\![\Delta]\!]_{\mathsf{P}} \parr [\![\Psi]\!]_{\mathsf{P}} \parr (A^{\perp} \otimes A)}}{[\![\Delta]\!]_{\mathsf{P}} \parr [\![\Psi]\!]_{\mathsf{P}}}$$

This procedure allows us to build the required proof of $[\![\Gamma]\!]_{\mathsf{P}}$, although all proofs built this way have a particular shape, reflecting the shape of the original proof tree in the sequent calculus. □

Remark 1.11. *None of the rules of the up fragment is needed to prove completeness, except the cut rule* $\mathsf{i}\uparrow$*, so that they are admissible in* $\mathsf{LS} \cup \{\mathsf{i}\uparrow\}$*. Moreover, the* $\mathsf{i}\uparrow$ *rule is used only to translate the cut rule* cut *of the sequent calculus. Therefore, by using the cut elimination result of the* $\mathsf{LL} \cup \{\mathsf{cut}\}$ *calculus, we can deduce the completeness of* LS*. This is an external cut admissibility result for* SLS*.*

1.3 From Equations to Inference Rules

The SLS system we have described allows for a natural representation of proofs of linear logic, comparable to the $\mathsf{LL} \cup \{\mathsf{cut}\}$ sequent calculus — with some benefits, since it is made simpler by the use of a congruence and has for example a *functorial* promotion rule, which is local in its application. However, we are at least partially concerned here with the use of this system for proof search, and therefore we need to make the proof construction process as explicit as possible. In the deep inference methodology, proof search is performed by rewriting of structures and not simple formulas, so that it must take into account the congruence.

To stay close to the proof search perspective, we define a variant of SLS where all equations are removed, so that structures are plain formulas, and inference rules are added to keep completeness by performing operations that were handled by the congruence. The inference rules of the system, called SLSE, are shown in Figure 4 — it is mostly a superset of the rules of SLS. The inference rules introduced in SLSE are $\mathsf{e}\downarrow$, $\mathsf{b}\downarrow$, $\mathsf{j}\downarrow$, $\alpha\downarrow$, $\sigma\downarrow$ and $\mathsf{v}\downarrow$, as well as their duals in the up fragment. However, this is not the only difference with SLS, since some rules have been modified, in comparison of the original system:

- There are two switch rules, called s_L and s_R which are self-dual and have been separated because there is no rule for the commutativity of the $\otimes$ connective.
- The rule $\mathsf{o}\downarrow$ handling $\oplus$ connectives is splitted into the two rules $\mathsf{o}_L\downarrow$ and $\mathsf{o}_R\downarrow$, since there is no rule for the commutativity of $\oplus$ — the same change is done in the up fragment, on the $\mathsf{o}\uparrow$ rule.

In order to make sure that the rules introduced to replace equations are enough to retain completeness of the system, we prove that all equations used in SLS can be simulated by inference rules that are admissible in SLSE.

Definition 1.12. *An equation* $A \equiv B$ *is said to be* admissible *in a proof system if both the rule with premise* A *and conclusion* B *and its dual, are admissible in this system.*

For each equation used in SLS, we now prove that the two corresponding rules are admissible in LSE, by showing that if there is a proof of one formula, there is also a proof of the other formula. This means we have to prove two implications for each equation[2] used in SLS, but some of them are already implemented as rules in LSE, and most of the others are routine inductions on the height of a given proof. Unless stated otherwise, the following proofs can implicitly use several times their induction hypothesis, as the transformation preserves the height of proofs.

[2] Compared to the number of rules in the SLS system, this is quite a lot to verify, but the high number of inference rules to use or prove admissible in LSE is the price for being explicit, and it is well-known that » *the devil is in the detail* «.

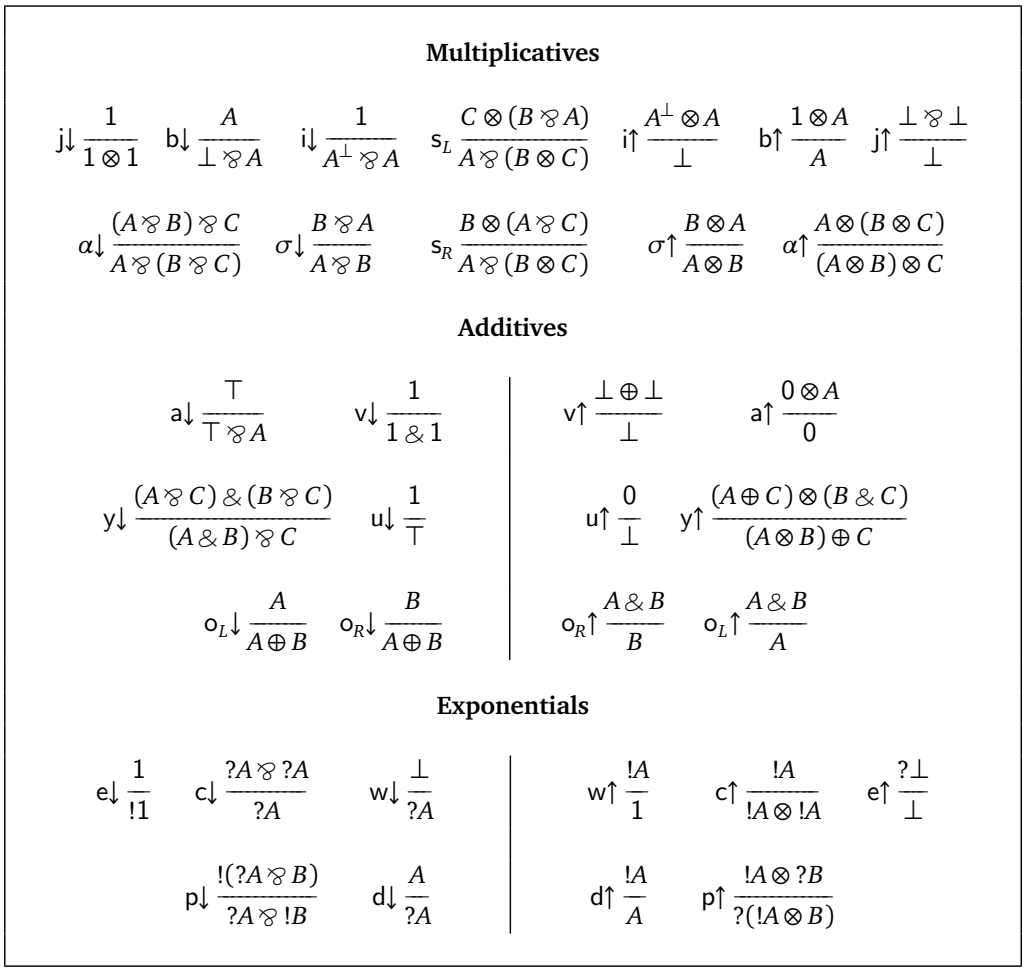


Figure 4: Inference rules for system SLSE

Lemma 1.13. *The equation $A \& B \equiv B \& A$ is admissible in* LSE.

Proof. The two rules corresponding to this equation are symmetric, so that we only have to prove that if there is a proof of some structure $\xi\{A \& B\}$ in LSE, then there is also a proof of $\xi\{B \& A\}$. We proceed by induction on the height of some proof $\mathscr{P}$ of $\xi\{A \& B\}$, using a case analysis on its bottommost rule instance r. In the base case, r is a v↓ instance and we are done. In the general case, if the structure $A \& B$ is not modified or if changes happen only inside A or B, we rewrite $A \& B$ into $B \& A$ in this instance and we conclude by induction hypothesis on the proof above. There is only one other case, when r is some instance of y↓ applied on $(A \& B) \parr C$. In this situation, we can rewrite $(A \parr C) \& (B \parr C)$ into $(B \parr C) \& (A \parr C)$ and conclude by induction hypothesis. □

Lemma 1.14. *The equation $A \parr (B \parr C) \equiv (A \parr B) \parr C$ is admissible in* LSE.

Proof. One of the rules corresponding to this equation is already in the LSE system, it is the $\alpha\downarrow$ rule. The other rule can be obtained from $\alpha\downarrow$ by commutativity, so that it is not only admissible but derivable, as follows:

$$\dfrac{\dfrac{\dfrac{\xi\{A \parr (B \parr C)\}}{\xi\{(C \parr B) \parr A\}}\,\sigma\downarrow^*}{\xi\{C \parr (B \parr A)\}}\,\alpha\downarrow}{\xi\{(A \parr B) \parr C\}}\,\sigma\downarrow^*$$

□

Lemma 1.15. *The equation $\bot \parr A \equiv A$ is admissible in* LSE.

Proof. The $\mathsf{b}\downarrow$ rule in LSE corresponds to one of the directions of this equation. In the other direction, we need to prove that if there is a proof of $\xi\{\bot \parr A\}$, then there is a proof of $\xi\{A\}$, in LSE. Again, we proceed by induction on the height of a proof $\mathscr{P}$ of $\xi\{A \parr \bot\}$ or $\xi\{\bot \parr A\}$. Most cases are handled by simply rewriting the formula and using the induction hypothesis, and we are done when the structure is erased or when the $\mathsf{b}\downarrow$ rule is encountered. □

Lemma 1.16. *The equation $\top \with A \equiv A$ is admissible in* LSE.

Proof. For this equation, we need to prove that both of the corresponding rules are admissible. Given a proof $\mathscr{P}$ of $\xi\{A\}$ in one direction and $\xi\{\top \with A\}$ in the other, we use an induction on the pair (c, h) under lexicographic order, where c is the number of $\mathsf{y}\downarrow$ and $\mathsf{c}\downarrow$ instances in $\mathscr{P}$ and h is the height of $\mathscr{P}$. In the first direction, we can rewrite $\xi\{A\}$ as $\xi\{\top \with A\}$ and use the induction hypothesis in most cases, when the bottommost instance r does not affect A, or affects only substructures of A. In the case of an interaction between A and its context, we introduce new rule instances:

$$\mathsf{s}_L\dfrac{\xi\{C \otimes (B \parr D)\}}{\xi\{D \parr (B \otimes C)\}} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{\xi\{\top \with (C \otimes (B \parr D))\}}{\xi\{\top \with (D \parr (B \otimes C))\}}\,\mathsf{s}_L}{\xi\{\top \with ((B \otimes C) \parr D)\}}\,\sigma\downarrow}{\xi\{(\top \parr D) \with ((B \otimes C) \parr D)\}}\,\mathsf{a}\downarrow}{\xi\{(\top \with (B \otimes C)) \parr D\}}\,\mathsf{y}\downarrow\ \dfrac{}{\xi\{D \parr (\top \with (B \otimes C))\}}\,\sigma\downarrow$$

For this transformation, we are done when A is 1 and we can introduce a derivation from 1 to $\top \with 1$ instead, formed by a $\mathsf{u}\downarrow$ instance and a $\mathsf{v}\downarrow$ instance, as follows:

$$\xi\{1\} \quad\longrightarrow\quad \dfrac{\dfrac{\xi\{1\}}{\xi\{1 \with 1\}}\,\mathsf{v}\downarrow}{\xi\{\top \with 1\}}\,\mathsf{u}\downarrow$$

In the other direction, we prove the more general result that from any given proof of $\xi\{B \with A\}$ we can build a proof of $\xi\{A\}$. In the base case, both A and B are 1 and the result is trivial. In the general case, we always use the induction hypothesis. □

Lemma 1.17. *The equation* $?\bot \equiv \bot$ *is admissible in* LSE.

Proof. One of the rules corresponding to this equation is the dereliction $\mathsf{d}\downarrow$, already present in LSE. In the other direction, we consider a proof $\mathscr{P}$ of $\xi\{?\bot\}$, and show by induction on the pair (c,h), where c is the number of $\mathsf{y}\downarrow$ and $\mathsf{c}\downarrow$ instances in $\mathscr{P}$ and h is the height of $\mathscr{P}$, how to build a proof of $\xi\{\bot\}$. In the base cases, the bottommost rule instance in $\mathscr{D}$ is a weakening $\mathsf{w}\downarrow$ or a dereliction $\mathsf{d}\downarrow$ affecting $?\bot$, we are done. In most other cases, we use the induction hypothesis. In the case of a promotion $\mathsf{p}\downarrow$ applied on $?\bot$, Lemma 1.15 allows us to use the equation $\bot \parr B \equiv B$ as follows:

$$\mathsf{p}\downarrow\dfrac{\xi\{!(?\bot \parr B)\}}{\xi\{?\bot \parr !B\}} \qquad \longrightarrow \qquad \dfrac{\dfrac{\xi\{!(\bot \parr B)\}}{\xi\{!B\}}\equiv}{\xi\{\bot \parr !B\}}\mathsf{b}\downarrow$$

and the case of a contraction $\mathsf{c}\downarrow$ applied on $?\bot$ is treated similarly, by replacing the contraction with a rule corresponding to the use of Lemma 1.15. □

Lemma 1.18. *The equation* $1 \with 1 \equiv 1$ *is admissible in* LSE.

Proof. The $\mathsf{v}\downarrow$ rule in LSE corresponds to one direction of this equation. In the other direction we prove that for any proof $\mathscr{P}$ of $\xi\{1 \with 1\}$, we can build a proof of $\xi\{1\}$ using the equation $\top \with A \equiv A$, admissible by Lemma 1.16. This proof construction is direct, as follows:

$$\begin{array}{c}\vdots\\ \xi\{1 \with 1\}\end{array} \qquad \longrightarrow \qquad \dfrac{\dfrac{\begin{array}{c}\vdots\\ \xi\{1 \with 1\}\end{array}}{\xi\{\top \with 1\}}\mathsf{u}\downarrow}{\xi\{1\}}\equiv$$

□

In order to prove admissible the rest of the equations, we need another lemma, symmetric to the admissibility of a part of the equation $A \parr \bot \equiv A$. This is the only part of up fragment equations — the ones shown on the right in Figure 1 — that we need to show admissible directly, rather than through admissibility of its dual.

Lemma 1.19. *If there is a proof of* $\xi\{A\}$ *in* LSE*, there is also a proof of* $\xi\{1 \otimes A\}$.

Proof. We proceed by induction on the pair (c,h), defined as previously for a proof $\mathscr{D}$ of $\xi\{A\}$, to show a stronger result: we can build a proof of $\xi\{A \otimes 1\}$ and a proof of $\xi\{1 \otimes A\}$ in LSE. In the base case, A is 1 and we use an instance of the $\mathsf{j}\downarrow$ rule, for both proofs. All other cases rely just on the induction hypothesis, except those where A interacts with its context. In such cases, we introduce a switch instance to move the material to A, and use the other induction hypothesis — symmetric to the current one, since the switch swaps the positions of the 1 and the A. □

Now, we can use this lemma to build derivations in SLSE, involving a cut, that are equivalent to the inference rules induced by the other equations of SLS, dual to the equations that we have already shown admissible. The idea is that the *detour* created by a pair of identity and cut instances allows to manipulate the dual of the target formula rather than the formula itself.

Proposition 1.20. *If an inference rule with premise A and conclusion B is admissible in* LSE*, then its dual with premise $B^\perp$ and conclusion $A^\perp$ is admissible in* $\mathsf{LSE} \cup \{\mathsf{i}\uparrow\}$.

Proof. This is a standard result in deep inference, obtained by building a derivation corresponding to the dual of the rule, using a cut and identity pair to flip its premise and conclusion. However, in this setting, the proof is slightly more complicated. We need to show that if there is a proof of $\xi\{B^\perp\}$ in $\mathsf{LSE} \cup \{\mathsf{i}\uparrow\}$, there is also a proof of $\xi\{A^\perp\}$ in this system. Now, by Lemma 1.19 we can build a proof $\mathscr{D}$ of $\xi\{1 \otimes B^\perp\}$ from the given proof of $\xi\{B^\perp\}$ and use it to build the expected proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\xi\{1 \otimes B^\perp\}}{\xi\{B^\perp \otimes 1\}}\equiv}{\xi\{B^\perp \otimes (A \parr A^\perp)\}}\mathsf{i}\downarrow}{\xi\{A^\perp \parr (A \otimes B^\perp)\}}\mathsf{s}_L}{\xi\{A^\perp \parr (B \otimes B^\perp)\}}}{\xi\{A^\perp \parr \bot\}}\mathsf{i}\uparrow}{\xi\{A^\perp\}}\equiv^*$$

□

Finally, we can reach the final conclusion, that the SLSE system is sound and complete with respect to linear logic, since it can simulate any proof of SLS, which is itself complete. Because these systems are almost the same, this boils down to the ability of simulating all equations of SLS, which follows from the previous lemmas.

Theorem 1.21. *For any formula A of linear logic, A is provable in* SLSE *if and only if the corresponding structure A is provable in* SLS.

Proof. In the first direction, the result is trivial, since from any proof of A in SLSE we can build a proof of A in SLS by replacement of the extra rules added to simulate equations by an application of the congruence, and adjusting the use of the s_L and s_R rules — equivalent to s through the congruence — and of the $\mathsf{o}_L\downarrow$ and $\mathsf{o}_R\downarrow$ rules — equivalent to $\mathsf{o}\downarrow$ by commutativity of the $\oplus$ connective.

In the other direction, we build a proof of A in SLSE from a proof of A in SLS by explicitly using the $\equiv$ rule in the given proof in SLS, and then apply Lemma 1.13, Lemma 1.14, Lemma 1.15, Lemma 1.16, Lemma 1.17 and Lemma 1.18, as well as Proposition 1.20 to rewrite it into a valid proof of A in SLSE. □

From the viewpoint of the design of deductive systems, the SLSE system without equations is interesting because it shows that many of the equations which are used through the congruence present in the calculus of structures are either completely superfluous, or could be used in only one direction. This parcimony in the design of a system is not only useful from the viewpoint of an implementation, but also provides more insight on the structure of proofs. This observation can be applied in general to the calculus of structures, as systems such as SKS could be modified to avoid using equations. Notice that this also questions the definitions of the sequent structure in the sequent calculus, where equations simulated by the use of either sets or multisets.

2 Systems with Explicit Polarities

The notion of *polarity* is an important byproduct of the fine-grained study of logical systems induced by linear logic, such as the analysis of classical logic [Gir91] and of the translations and computational interpretations of classical and intuitionistic systems [Lau02], although it was already implicitly present in existing systems for intuitionistic logic. There is a strong connection between polarities, as assigned to the connectives and to formulas, and the properties of the corresponding inference rules in the sequent calculus. In particular, this notion is a key to the development of normal forms known as *focused proofs* in linear logic [And92], which are useful to build proof search procedures and logic programming languages.

For this reason, we consider the systems where polarities appear explicitly at the level of formulas, and in particular we refine the LSE system defined previously, to follow this methodology. The introduction of polarities will be an important step in the definition of the normal forms we are ultimately interested in.

2.1 Polarised Formulas and Calculi

The starting point in the introduction of polarities is the redefinition of formulas, to which polarities are assigned. This creates two syntactic categories, the positive and the negative formulas, that respect negation in the sense that the dual of some positive formula is a negative formula, and conversely. Historically, these categories have been defined to reflect the behaviour of the corresponding inference rules in the sequent calculus, but one can also consider them as the only distribution such that duality switches from one group to the other, and each group contains both a conjuntion and a disjunction, with their respective units, at least when considering the fragment without exponentials — the polarity of exponentials is less clear, and this reflects the complexity of these connectives, compared to the others.

Definition 2.1. *The* polarised formulas *of linear logic are defined in two categories,* positive *formulas defined by P and* negative *formulas defined by N in the grammar:*

$$\begin{aligned} P,Q &::= a \mid 1 \mid P \otimes Q \mid 0 \mid P \oplus Q \mid !N \mid \downarrow N \\ N,M &::= \overline{a} \mid \bot \mid N \parr M \mid \top \mid N \mathbin{\&} M \mid ?P \mid \uparrow P \end{aligned}$$

Note that there are explicit shifts in this syntax, so that a positive formula can appear within a negative one, using a $\uparrow$ shift, and the other way around — this is not the case in LLP [Lau02], where shift is only performed by exponentials. Then, standard formulas can easily be translated into the polarised setting.

Definition 2.2. *The* polarised translation $\lfloor A \rfloor$ *of a formula A of linear logic is defined on the structure of A such that $\lfloor A \rfloor = \lfloor A \rfloor^+$, with the following mutual inductions:*

$$\begin{array}{llll} \lfloor 1 \rfloor^+ = 1 & \lfloor A \otimes B \rfloor^+ = \lfloor A \rfloor^+ \otimes \lfloor B \rfloor^+ & \lfloor \bot \rfloor^- = \bot & \lfloor A \parr B \rfloor^- = \lfloor A \rfloor^- \parr \lfloor B \rfloor^- \\ \lfloor 0 \rfloor^+ = 0 & \lfloor A \oplus B \rfloor^+ = \lfloor A \rfloor^+ \oplus \lfloor B \rfloor^+ & \lfloor \top \rfloor^- = \top & \lfloor A \mathbin{\&} B \rfloor^- = \lfloor A \rfloor^- \mathbin{\&} \lfloor B \rfloor^- \\ \lfloor a \rfloor^+ = a & \lfloor !A \rfloor^+ = !\lfloor A \rfloor^- & \lfloor \overline{a} \rfloor^- = \overline{a} & \lfloor ?A \rfloor^- = ?\lfloor A \rfloor^+ \end{array}$$

and $\lfloor A \rfloor^+ = \downarrow \lfloor A \rfloor^-$ otherwise *and $\lfloor A \rfloor^- = \uparrow \lfloor A \rfloor^+$ otherwise*

The other way around, the *unpolarised translation* $\lceil A \rceil$ of a polarised formula A is simply defined by removing all shifts $\uparrow$ and $\downarrow$ in A. Moreover, we call *minimally polarised* a formula A which contains no more shifts than $\lfloor \lceil A \rceil \rfloor$, or equivalently a formula which contains no shift pair $\uparrow\downarrow$ or $\downarrow\uparrow$.

The distinction between positive and negative formulas creates distinctions at all levels of the system, and in particular this raises the question of the polarity of the conclusion of a derivations and proofs. In the sequent calculus, the conclusion is a sequent where the comma corresponds to $\parr$, so that it is always considered as negative. But in the calculus of structures, we may need to plug a derivation in any given context, so that we have to handle both cases. For the sake of simplicity, we will consider both positive and negative derivations, but only positive proofs, using the following manipulation of formulas.

Definition 2.3. *The* positivation A^+ *of a polarised formula A is defined as A when the formula A is positive, and $\downarrow A$ when it is negative.*

In the following, a proof of A will implicitly refer to a proof of A^+. Notice that from such a proof, a negative derivation can be built by adding a shift in front of the toplevel structures in every rule instances. Then, the contexts need to be able to distinguish between polarities.

Definition 2.4. *A polarised context ξ is said to be* positive *or* negative *when the result of inserting a polarised formula A in place of its hole is a valid polarised formula only if A is positive or negative, respectively.*

If the resulting formula $\xi\{A\}$ is positive, the context ξ is said to be *outer-positive*, and if it is negative then ξ is an *outer-negative* context. Usually, only *homogeneous* contexts will be used — these are the ones where the hole has the same polarity as the whole resulting formula.

Polarity of rules and connectives. Traditionally, polarities are assigned to the connectives, and thus formulas, of the logic. However, as mentioned in Chapter 2, the use of polarities is mainly related to the behaviour of inference rule instances, and in particular to their permutability properties. This observation, which can be illustrated by the ambiguous handling of polarities for exponentials, in the sequent calculus, is even more important in the calculus of structures where the rules are decomposed and often describe an interaction between two connectives rather than the decomposition of a connective. For example, the switch rule:

$$\mathsf{s}_L \frac{(A \parr B) \otimes C}{A \parr (B \otimes C)}$$

corresponds to the operation of context-splitting making the $\otimes$ rule non-invertible in the sequent calculus, so that it is clearly a positive — or synchronous — rule, but it involves a $\parr$, which is a negative connective. This can seem disturbing, since the $\parr$ structure is modified just as much as the $\otimes$ in this rewriting, but the point is that this operation is intrinsically synchronous. Here, we avoided another disturbing example by using the g rule for $\&$ rather than the distribution rule which makes $\&$ interact with a $\oplus$ [Str03a].

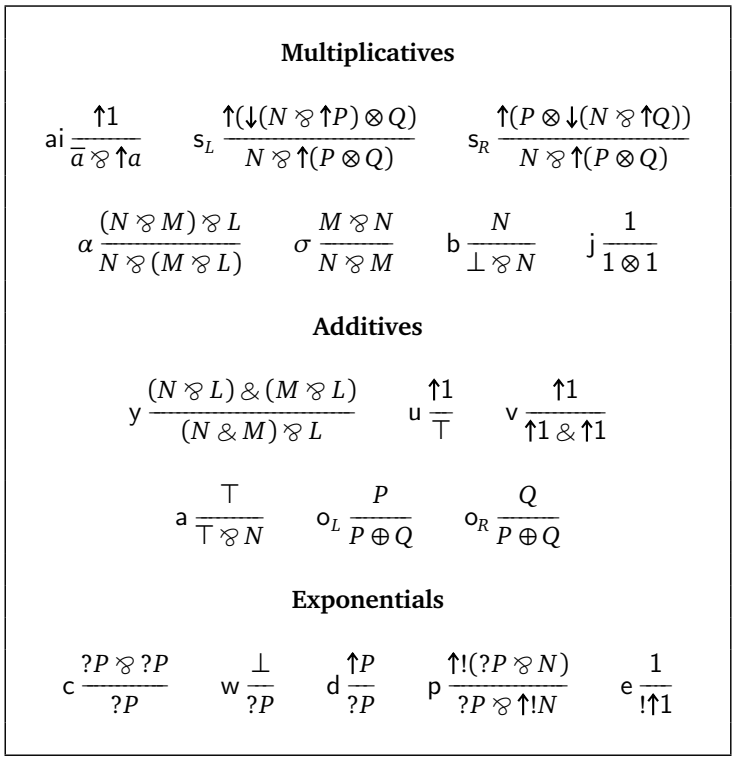


Figure 5: Inference rules for system LEP°

2.2 The Polarised System LEP

The first step of the refinement of the LSE proof system is to make polarities explicit in formulas, and translate directly all inference rules into this polarised setting. We obtain the LEP° system shown in Figure 5, by introducing the polarity annotations everywhere in inference rules. Also, the switch rule s_L used here is a variation of the $s_L\downarrow$ rule used in SLSE, which is not self-dual but is better suited[3] for our proof search purposes — this is not a problem, as we will not concentrate on the dual, up fragment of the system in this section. Notice that because of explicit shifts, even the other switch rule s_R is not self-dual.

Remark 2.5. *In the presence of polarity annotations, the plugging of formulas inside contexts must respect the additional conditions of well-formedness applied to polarised formulas, and inference rules as well. In the* LEP° *system, the polarity of the premise of a rule is always the same as the polarity of its conclusion.*

The LEP° system is essentially the same as the LSE system, and it is interesting to remark that most of the rules are naturally compatible with polarities.

[3]Keeping the left switch rule s_L of the SLS system would have been problematic regarding the use of commutativity rules, making derivations more difficult to read.

A crucial observation concerning the LEP° system is that polarity shifts are not only introduced in a formula which is then proved, but also introduced during the proof search process. In particular, the switch rules s_L and s_R are responsible for the accumulation of shifts during proof search, and we need a way to clean up these annotations — since they can block the further application of inference rules. For this reason, we complete the system with the rules:

$$\tau_+ \frac{P}{\downarrow\uparrow P} \qquad\qquad \tau_- \frac{N}{\uparrow\downarrow N}$$

to form the system $\mathsf{LEP} = \mathsf{LEP}^\circ \cup \{\tau_+, \tau_-\}$, a polarised system for linear logic that we will study and compare to the unpolarised LSE system.

Remark 2.6. *Although the polarity shifts appearing in a formula during proof search have no immediate logical meaning, they syntactically prevent the system from being complete without the cleaning τ_+ and τ_- rules. For instance, the polarised formula $\uparrow\downarrow\overline{a} \parr a$ has only one proof in LEP, which critically uses theses rules, as shown below:*

$$\tau_- \dfrac{\mathsf{i}\dfrac{\tau_+ \dfrac{1}{\downarrow\uparrow 1}}{\downarrow(\overline{a} \parr a)}}{\downarrow(\uparrow\downarrow\overline{a} \parr a)}$$

In order to establish a precise correspondence between LEP and LSE, we need to fill the gap created by the variant rules introduced in LEP. This is easy, as we can simply express one version of a rule in the other system. In the case of the switch rule s_L, this is just a matter of reorganisation[4] of the formula.

Lemma 2.7. *The version of the switch s_L used in LEP, when depolarised, is admissible in LSE, and the version used in LSE, when polarised, is admissible in LEP.*

Proof. In the LSE system, we can easily obtain the other version of s_L by using both commutativity rules in the up and down fragments, as shown below on the left, and then produce the required proof by eliminating the $\sigma\uparrow$ instance, by admissibility of the up fragment. In LEP, we can use the σ rule, as shown below on the right, and the commutativity of $\otimes$, which can be shown admissible by a simple induction:

$$\dfrac{\dfrac{\dfrac{\xi\{(A \parr B) \otimes C\}}{\xi\{(B \parr A) \otimes C\}}\sigma\downarrow}{\xi\{C \otimes (B \parr A)\}}\sigma\uparrow}{\xi\{A \parr (B \otimes C)\}}\mathsf{s}_L \qquad\qquad \dfrac{\dfrac{\dfrac{\xi\{\uparrow(Q \otimes \downarrow(\uparrow P \parr N))\}}{\xi\{\uparrow(Q \otimes \downarrow(N \parr \uparrow P))\}}\sigma}{\xi\{\uparrow(\downarrow(N \parr \uparrow P) \otimes Q)\}}\sigma}{\xi\{N \parr \uparrow(P \otimes Q)\}}\mathsf{s}_L$$

and use this derivation as a replacement for the original s_L rule of the LSE system, after introduction of the polarities. □

[4] There is no form of switch that could satisfy the criterions of all the different systems used in this section, but the transformation of one form into another is made easy by the commutativity rule.

The polarised syntax we use allows to compose freely the two shift operators, so that there is no reason, in general, to consider only the polarised formulas that are minimal, in the sense that there is an unpolarised formula A such that $\lfloor A \rfloor$ is the considered formula. However, some rules of LEP apply only on formulas which are locally minimal. We can overcome this problem by removing unnecessary shifts.

Lemma 2.8. *For any polarised formula A of linear logic, there is a derivation from the minimally polarised formula $\lfloor \lceil A \rceil \rfloor$ to A in* LEP.

Proof. We proceed by induction on the number of shift operators in A. In the base case, the formula A is minimally polarised, so that $A = \lfloor \lceil A \rceil \rfloor$. Otherwise, there is a subformula of the shape $\downarrow\uparrow P$ or $\uparrow\downarrow N$ inside A. We can thus remove these two shifts and use the induction hypothesis, and then complete the resulting derivation either with a τ_+ instance or a τ_- instance. □

We can now prove that the system LEP is equivalent to the unpolarised system LSE, by translating proofs from one system to the other. In one direction, this is just a matter of rewriting formulas, and in the other it requires to replace rules with derivations produced by the previous lemma.

Theorem 2.9. *For any polarised formula A of linear logic, A is provable in* LEP *if and only if the unpolarised formula $\lceil A \rceil$ is provable in* LSE.

Proof. Given a proof $\mathscr{P}$ of A in the polarised system LEP, we can produce a proof of $\lceil A \rceil$ in the unpolarised system LSE by removing shifts from formulas in $\mathscr{P}$, and replacing instances of LEP rules by instances of the corresponding rules in LSE — using Lemma 2.7 to handle the s_L rule.

In the other direction, we proceed by structural induction on the given proof $\mathscr{P}$ of $\lceil A \rceil$, to produce the proof of A. At each step, we can use the induction hypothesis and the rule of LEP corresponding to the encountered rule instance of LSE, after the transformation of A into $\lfloor \lceil A \rceil \rfloor$ by Lemma 2.8. Applying the corresponding LEP rule is always possible because the conclusions of all these rules match minimally polarised formulas. □

We can also prove that the logical relation between A and $\lfloor \lceil A \rceil \rfloor$ is actually an equivalence, by showing that from a proof of any formula, a proof can be built for its minimally polarised version.

Lemma 2.10. *For any polarised formula A of linear logic, if A is provable in* LEP *then the minimally polarised formula $\lfloor \lceil A \rceil \rfloor$ in also provable in* LEP.

Proof. By induction on the height of a proof $\mathscr{P}$ of A in LEP, using a case analysis on the bottommost rule instance r in $\mathscr{P}$. In the base case, r is a δ_+ or δ_- instance that removes the last pair $\downarrow\uparrow$ or $\uparrow\downarrow$ from a formula, and we can use the proof above. In the general case, if r is a δ_+ or δ_- instance then it is removed and the induction hypothesis is used, while in all other cases, the induction hypothesis can be used immediately and r reused, since no rule in LEP can modifiy a pair of shifts — and the induction hypothesis can be used twice if required, since the resulting proof is at most of the same height as $\mathscr{P}$. □

3 From Polarities to Focusing

The system SLSE for linear logic, described in the previous section, is interesting from the point of view of proof search, because it is completely explicit — there are no equations used implicitly between inference rules applications, and the objects being rewritten are plain formulas. However, for other reasons there is still a huge amount of non-determinism in the proof search process in this system:

- The search is not organised by branches, as in the sequent calculus, and thus at each step one must choose in which part of the formula, corresponding to a branch, to apply the next inference rule.
- Among different formulas within the equivalent of a branch — that is, some sequence of formulas separated by ⅋ — the same non-determinism as in the sequent calculus is present, induced by the choice of the next formula to treat and the of the inference rule to apply.
- There are more rules in SLSE than in the sequent calculus, as we turned into rules equations that are handled implicitly in the sequent calculus, through the definition of sequents as multisets of formulas, and through branches.
- The SLSE system is symmetric, and the power of its up fragment corresponds to the power of the cut rule of the sequent calculus, so that we should restrict the system to the cut-free, down fragment LSE.

Our goal here is to improve the situation from the proof search perspective, by restricting this system until we obtain a proof search procedure guided by strong constraints, while retaining its completeness. The first step, as mentioned above, is to consider only the down fragment LSE, which is complete with respect to linear logic — this is obvious, as performing proof search in the presence of cut becomes extremely difficult, since the cut breaks the subformula property and thus requires a *» clever guess «* to be applied. Beyond this step, there is no obvious restriction that would not deprive us from too many proofs — an outermost-leftmost-first strategy would produce only proofs directly equivalent to sequent calculus proofs, and deny any benefit to the use of the deep inference methodology, for instance.

We need guidance to design such restrictions, and we will follow the concept of polarities to achieve this, with the slogan that *» although some proofs do not respect polarities, the set of proofs respecting them is enough to retain completeness «*. What we mean by *» respecting «* polarities here is that a given polarised formula should not require more ↑ or ↓ shifts than it contains originally to be proved. This is similar to the subformula property, in the sense that we want a system where any shift in the premise of a rule instance corresponds to a shift present in its conclusion. This idea leas us to a system which is more than just cut-free, because it respects this stronger form of the subformula property — although in terms of formulas, the property is not stated as in sequent calculi, since there is no meta-level. The system LEP is the natural starting point in this study, because it is close to the unpolarised system LSE but is equipped with the syntax necessary to state the focusing result.

3.1 The Focused System LEF

In the polarised system LEP, the borders between subformulas of different polarity are made explicit by the shift operators ↓ and ↑, but LEP does not respect polarities in the sense that it requires the introduction of new shifts in a formula during proof search, when applying the switch rules:

$$\mathsf{s}_L \frac{\uparrow(\downarrow(N \parr \uparrow P) \otimes Q)}{N \parr \uparrow(P \otimes Q)} \qquad \mathsf{s}_R \frac{\uparrow(P \otimes \downarrow(N \parr \uparrow Q))}{N \parr \uparrow(P \otimes Q)}$$

In order to design a system where polarities are respected — that means, not only a copy of LSE where polarities are made explicit, but a system constrained by polarities — we need to modify these rules. We restrict the use of the switch rules to a positive context, so that there is no need to introduce an additional ↑ shift, and obtain the following replacement rules:

$$\mathsf{s}_L \frac{\downarrow(N \parr \uparrow P) \otimes Q}{\downarrow(N \parr \uparrow(P \otimes Q))} \qquad \mathsf{s}_R \frac{P \otimes \downarrow(N \parr \uparrow Q)}{\downarrow(N \parr \uparrow(P \otimes Q))}$$

In these rules, which are of critical use in the proof search process, we can see the importance given to formulas of the shape $\downarrow(- \parr \uparrow -)$, since the s_L and s_R rules preserve this shape, which is modified only when the negative formula N has been *pushed* through a complete layer of ⊗ connectives. This is characteristic of a *positive focus phase* in standard focused sequent calculi [And92], formed of a sequence of rule instances dealing with a unique layer of positive connectives. In such a sequent calculus, a sequent of the shape:

$$\vdash \Gamma, [P] \qquad \text{where } [P] \text{ means » } P \textit{ under focus } \text{«}$$

is the conclusion of a proof where a synthetic positive formula P is decomposed by interaction with the formulas in Γ, to produce premises where negative formulas are decomposed to obtain sequents where one positive formula can be put under focus. In the calculus of structures, no *decomposition* of P really happens, and the interaction of Γ and P is seen as the aggregation of several steps, through the use of several switches. Therefore, we can consider the switch rules described above as the interaction of one negative[5] formula with a positive formula.

Remark 3.1. *In the sequent calculus, the primary difference between a focused and an unfocused system is the* vertical grouping *of positive rule instances, but in any sequent calculus, one can see some* horizontal grouping *of interactions between formulas in a branching rule instance, when looking through the glasses of deep inference where switches allow to consider the equivalent non-grouped interaction. It is thus legitimate that horizontal grouping is not enforced in a focused calculus of structures, as this kind of grouping is an artifact of the sequent calculus and not an essential byproduct of the focused normal form.*

[5] In the usual focused sequent calculus, all the formulas in the sequent $\vdash \Gamma, [P]$ should be considered as negative formulas, in the sense that comma is a ⅋, so that if the system was polarised it would require shifts to appear on P positive formulas in Γ.

Restrictions on LEP. The comparison between focused sequent calculi and the polarised LEP system provides some insight on the use of the shifts: any formula of the shape $\uparrow P$ can be seen as a positive $[P]$ under focus, which can interact with a negative formula N if both of them are inserted under a shift, thus forming the conclusion $\downarrow(N \parr \uparrow P)$ of the modified switches. This can be internalised by defining the new connective $\oplus$ that extends the polarised formulas into potentially focused polarised formulas, with the following intended meaning:

$$N \oplus P \quad \equiv \quad \downarrow(N \parr \uparrow P)$$

This new connective describes a local variant of the sequent $\vdash \Gamma, [P]$, and it has the particularity of accepting only subformulas of different polarity, one negative and one positive. As a consequence, we use it as a non-commutative connective, to keep the distinction between the two subformulas easy to read. This extension of the language of formulas[6] is the first step on our way to the definition of a focused calculus of structures.

Definition 3.2. *A* simply focused formula *is a polarised formula A which contains exactly one subformula of the shape $N \oplus P$, and a* multi-focused formula *is a polarised formula containing more than one subformula of this shape.*

Notice that the language of formulas extended with this *» focusing connective «* naturally tends to support the idea of *multi-focusing* [CMS08], and even a further form of focusing where focused formulas can contain focused formulas interacting with negative subformulas. Following the standard approach to focusing, we can also adapt the observation that negative formulas can be completely *» treated «* — in the sequent calculus, that would mean decomposed — before they interact with positive formulas. To achieve this, we restrict further the use of switches, to avoid applying them on negative formulas where the toplevel connective is a $\parr$ or a $\&$, by defining new classes of formulas, that can be reflected on unpolarised formulas.

Definition 3.3. *The* active *and* reactive *formulas of linear logic, denoted below by F and U respectively, are defined by the following grammars:*

$$F \ ::= \ !N \mid \downarrow N \mid a \qquad\qquad U \ ::= \ ?P \mid \uparrow P \mid \overline{a}$$

Definition 3.4. *A formula A of linear logic is said to be* pre-reactive *if and only if there exists a polarised, reactive formula U such that $A = \lceil U \rceil$.*

Finally, the switch rules we will use in the definition of a focused system are the following restricted variant of the basic rules:

$$\mathsf{s}_L \frac{\downarrow(U \parr \uparrow P) \otimes Q}{\downarrow(U \parr \uparrow(P \otimes Q))} \qquad\qquad \mathsf{s}_R \frac{P \otimes \downarrow(U \parr \uparrow Q)}{\downarrow(U \parr \uparrow(P \otimes Q))} \tag{10}$$

which can also be expressed in terms of the new $\oplus$ connective. However, we start with the standard notation and we will translate these rules in the new syntax when the focused system is really defined.

[6] This methodology of introducing a new connective in the language is similar to the nested sequents approach to modal logics [Brü10], where the syntax is extended with meta-level equivalents of modal connectives, thus allowing to stay within the modal language rather than rely on annotations or labels.

In the process of modifying the handling of shifts in the switch rules, we have created a new problem. Indeed, the application of a switch now requires a formula of a certain shape, but other formulas can be written that should be provable — for example, it is impossible to prove $\overline{a} \parr \uparrow a$ with these switch rules. To regain the ability to prove such formulas, we need the following rules:

$$\delta_+ \frac{\downarrow\uparrow P}{P} \qquad \delta_- \frac{\uparrow\downarrow N}{N}$$

which are the opposite of the τ_+ and τ_- rules, since they introduce a delay during proof search. The restricted system obtained by addition of these rules is the last step before the focused calculus.

Definition 3.5. *The* LER *system is a variant of* LEP *where switches are replaced with the restrictions shown in* (10)*, and extended with the δ_+ and δ_- rules.*

The LER system uses restricted rules, but has four new rules allowing to handle shifts. We can now prove that the new rules provide enough ways of manipulating formulas to make the system complete with respect to LEP. We start with a result stating the invertibility of the rules dealing with the negative connecties $\parr$ and $\&$.

Lemma 3.6. *The following rules are admissible in the* LER *system:*

$$\overline{\mathsf{b}} \frac{\bot \parr N}{N} \qquad \overline{\mathsf{a}} \frac{\top \parr N}{\top} \qquad \overline{\mathsf{y}} \frac{(N \& M) \parr L}{(N \parr L) \& (M \parr L)}$$

Proof. For each of these rules, we proceed by induction on the height of a given proof $\mathscr{P}$ of the premise, and show how it can be transformed into a proof of the conclusion — and in all cases, the transformation is at most height-preserving:

1. For $\overline{\mathsf{b}}$, the induction hypothesis is extended to $N \parr \bot$, with a base case when b is used at the bottom of $\mathscr{P}$. In all other cases, the induction hypothesis is directly used and the bottommost rule instance reused, or removed if it was affecting this $\bot$ — note that a switch cannot be used to move $\bot$ inside N, since we use a restricted form of switch, and if a pair $\uparrow\downarrow$ is added on $\bot$ by δ_-, nothing can happen except the removal of this pair, so that the corresponding instance of τ_- can be removed immediately from the proof.

2. For $\overline{\mathsf{a}}$, the induction hypothesis is extended to $N \parr \top$, with a base case when a is used at the bottom of $\mathscr{P}$, and other cases are treated similarly to the cases used in the induction for $\overline{\mathsf{b}}$.

3. The case of $\overline{\mathsf{y}}$ is treated the same way, with an induction hypothesis extended to $L \parr (N \& M)$ and a base case when y is used at the bottom of $\mathscr{P}$.

Notice that in each of these inductions, the hypothesis is extended to deal with the commutativity rule σ, and the induction hypothesis might be applied twice in cases where the considered formula is duplicated by a c or y instance. □

Then, we need to prove the admissibility of two rules that will be used in the transformation of proofs in LEP into proofs in LER.

Lemma 3.7. *The following rules are admissible in the* LER *system:*

$$\mathsf{z}_1\,\frac{\downarrow(N \,\&\, M) \otimes P}{\downarrow(\uparrow(\downarrow N \otimes P) \,\&\, \uparrow(\downarrow M \otimes P))} \qquad \mathsf{z}_2\,\frac{\downarrow\top \otimes A}{\downarrow\top}$$

Proof. For each of these two rules, given a proof $\mathscr{P}$ of the premise, we proceed by induction on the pair (c, h), where c is the number of y and c instances in $\mathscr{P}$ and h is the height of $\mathscr{P}$. At each step, we use a case analysis on the bottommost instance in $\mathscr{P}$. In the case of z_1, we can always directly use the induction hypothesis, possibly twice when a y or c instance is encountered, except when a switch s_L or s_R affecting the considered formula is encountered. In these cases, we use the transformation of the instance:

$$\mathsf{s}_R\,\frac{\xi\{\downarrow(N \,\&\, M) \otimes \downarrow(U \parr \uparrow P)\}}{\xi\{\downarrow(U \parr \uparrow(\downarrow(N \,\&\, M) \otimes P))\}}$$

into the derivation:

$$\begin{array}{c}
\mathsf{s}_R;\mathsf{s}_R\,\dfrac{\xi\{\downarrow(\uparrow(\downarrow N \otimes \downarrow(U \parr \uparrow P)) \,\&\, \uparrow(\downarrow M \otimes \downarrow(U \parr \uparrow P)))\}}{\xi\{\downarrow(\uparrow\downarrow(U \parr \uparrow(\downarrow N \otimes P)) \,\&\, \uparrow\downarrow(U \parr \uparrow(\downarrow M \otimes P)))\}} \\
\delta_-;\delta_-\,\dfrac{\xi\{\downarrow((U \parr \uparrow(\downarrow N \otimes P)) \,\&\, (U \parr \uparrow(\downarrow M \otimes P)))\}}{\mathsf{y}\,\dfrac{\xi\{\downarrow(U \parr (\uparrow(\downarrow N \otimes P) \,\&\, \uparrow(\downarrow M \otimes P)))\}}{\tau_-\,\dfrac{\xi\{\downarrow(U \parr \uparrow\downarrow(\uparrow(\downarrow N \otimes P) \,\&\, \uparrow(\downarrow M \otimes P)))\}}{}}}
\end{array}$$

for s_R, then using the induction hypothesis, and a similar one for s_L. In the case of z_2, the induction hypothesis can also be used, except when matching switches are encountered. The transformation used for a right switch is the following:

$$\mathsf{s}_R\,\frac{\xi\{\downarrow\top \otimes \downarrow(U \parr \uparrow P)\}}{\xi\{\downarrow(U \parr \uparrow(\downarrow\top \otimes P))\}} \quad\longrightarrow\quad \mathsf{a}\,\frac{\xi\{\downarrow\top\}}{\tau_-\,\dfrac{\xi\{\downarrow(U \parr \top)\}}{\xi\{\downarrow(U \parr \uparrow\downarrow\top)\}}}$$

where the induction hypothesis can be used to produce the proof of $\xi\{\downarrow\top\}$ needed. For a left switch, the proof of $\xi\{\downarrow(U \parr \uparrow\downarrow\top) \otimes P\}$ located above the switch can be transformed into a proof of $\xi\{\downarrow\top \otimes P\}$ of the same height through a straightforward induction, and using the invertibility of a described by Lemma 3.6. Finally, we can apply the induction hypothesis to obtain a proof of $\xi\{\downarrow\top\}$. □

Using these rules, we can show how to translate proofs between the restricted system LER and the basic polarised system LEP. The most complicated part of this is of course to show that the generic form of switches used in LEP can be simulated in LER, and this is where the admissible rules come into play.

Lemma 3.8. *A formula A is provable in* LER *if and only if it is provable in* LEP.

Proof. In one direction, the result is straightforward: given a proof $\mathscr{P}$ of A in LER, we build a proof of A in LEP by induction on the length of $\mathscr{P}$. Indeed, any instance of a rule of LER is valid in LEP, except instances of the δ_+ and τ_+ rules, but when such instances are encountered, they can be removed by applying Lemma 2.10. In the other direction, given a proof $\mathscr{P}$ of A in LEP, we also use an induction on the height of $\mathscr{P}$. In most cases, we can apply the induction hypothesis and reuse the bottommost rule instance in $\mathscr{P}$, since an instance of a rule of LEP is valid in LER, except for switches. In the case of a switch instance, we show how to replace it by a valid derivation in LER, by induction on the size $|N|$ of the negative formula N being pushed inside. In the base case, either this is a valid switch instance or N is $\bot$ or $\top$. The last two cases are handled by the following transformations, for $\bot$:

$$\mathsf{s}_L\dfrac{\xi\{\downarrow(\bot \parr \uparrow P)\otimes Q\}}{\xi\{\downarrow(\bot \parr \uparrow(P\otimes Q))\}} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{\xi\{\downarrow(\bot \parr \uparrow P)\otimes Q\}}{\xi\{\downarrow\uparrow P\otimes Q\}}\,\overline{\mathsf{b}}}{\xi\{P\otimes Q\}}\,\delta_+}{\xi\{\downarrow\uparrow(P\otimes Q)\}}\,\tau_+}{\xi\{\downarrow(\bot \parr \uparrow(P\otimes Q))\}}\,\mathsf{b}$$

and for $\top$:

$$\mathsf{s}_L\dfrac{\xi\{\downarrow(\top \parr \uparrow P)\otimes Q\}}{\xi\{\downarrow(\top \parr \uparrow(P\otimes Q))\}} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\xi\{\downarrow(\top \parr \uparrow P)\otimes Q\}}{\xi\{\downarrow\top\otimes Q\}}\,\overline{\mathsf{a}}}{\xi\{\downarrow\top\}}\,\mathsf{z}_2}{\xi\{\downarrow(\top \parr \uparrow(P\otimes Q))\}}\,\mathsf{a}$$

where the rules $\overline{\mathsf{a}}$ and $\overline{\mathsf{b}}$ are admissible in LER, by Lemma 3.6, and the rule z_2 can be used since it is admissible by Lemma 3.7. In the general case, N can have $\parr$ as toplevel connective, and it is handled with the following transformation:

$$\mathsf{s}_L\dfrac{\xi\{\downarrow((N \parr M) \parr \uparrow P)\otimes Q\}}{\xi\{\downarrow((N \parr M) \parr \uparrow(P\otimes Q))\}} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\xi\{\downarrow((N \parr M) \parr \uparrow P)\otimes Q\}}{\xi\{\downarrow(N \parr (M \parr \uparrow P))\otimes Q\}}\,\alpha}{\xi\{\downarrow(N \parr \uparrow\downarrow(M \parr \uparrow P))\otimes Q\}}\,\tau_-}{\xi\{\downarrow(N \parr \uparrow(\downarrow(M \parr \uparrow P)\otimes Q))\}}\,\mathsf{s}_L}{\xi\{\downarrow(N \parr \uparrow\downarrow(M \parr \uparrow(P\otimes Q)))\}}\,\mathsf{s}_L}{\xi\{\downarrow(N \parr (M \parr \uparrow(P\otimes Q)))\}}\,\delta_-}{\xi\{\downarrow((N \parr M) \parr \uparrow(P\otimes Q))\}}\,\alpha/\sigma$$

where the newly created s_L instances can be dealt with by induction hypothesis, and possibly rewritten into valid LER derivations. Notice that the two derivations obtained by induction hypothesis can be plugged into this derivation because of the deep inference feature of the calculus.

Then, N can also have $\&$ as toplevel connective, and this case is handled with the transformation of the instance:

$$\mathsf{s}_L \dfrac{\xi\{\downarrow((N \,\&\, M) \parr \uparrow P) \otimes Q\}}{\xi\{\downarrow((N \,\&\, M) \parr \uparrow(P \otimes Q))\}}$$

into the derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\xi\{\downarrow((N \,\&\, M) \parr \uparrow P) \otimes Q\}}{\xi\{\downarrow((N \parr \uparrow P) \,\&\, (M \parr \uparrow P)) \otimes Q\}}\,\overline{\mathsf{y}}}{\xi\{\downarrow(\uparrow\downarrow(N \parr \uparrow P) \,\&\, \uparrow\downarrow(M \parr \uparrow P)) \otimes Q\}}\,\tau_- ; \tau_-}{\xi\{\downarrow(\uparrow(\downarrow(N \parr \uparrow P) \otimes Q) \,\&\, \uparrow(\downarrow(M \parr \uparrow P) \otimes Q))\}}\,\mathsf{z}_1}{\xi\{\downarrow(\uparrow\downarrow(N \parr \uparrow(P \otimes Q)) \,\&\, \uparrow\downarrow(M \parr \uparrow(P \otimes Q)))\}}\,\mathsf{s}_L ; \mathsf{s}_L}{\xi\{\downarrow((N \parr \uparrow(P \otimes Q)) \,\&\, (M \parr \uparrow(P \otimes Q)))\}}\,\delta_- ; \delta_-}{\xi\{\downarrow((N \,\&\, M) \parr \uparrow(P \otimes Q))\}}\,\mathsf{y}$$

where the $\overline{\mathsf{y}}$ rule can be used since it is admissible by Lemma 3.6, and z_1 can also be used because it is admissible by Lemma 3.7. All the cases involving a right switch s_R instance are treated the same way as the left switch instances shown above.

Finally, the main induction hypothesis is used on the proof above the considered switch instance, and the result is glued to the derivation used as a replacement for the switch. □

Focused syntax. The actual focused system that we define in framework of the calculus of structures is called LEF, its inference rules being shown in Figure 6. It is essentially the same as LER, but it is based on focused formulas, where the new connective $\oplus$ can appear. More importantly, it uses some more restrictions:

- The rules τ_+ and δ_+ are removed from the system, except for the use of τ_+ on the positive unit 1, which is necessary.
- The rules τ_- and δ_- are restricted to particular situations, which involve the interaction between a negative and a positive, and are used under the names r and f which correspond respectively to the following situations:

$$\tau_- \dfrac{\downarrow(U \parr N)}{\downarrow(U \parr \uparrow\downarrow N)} \qquad\qquad \delta_- \dfrac{\uparrow\downarrow(U \parr \uparrow P)}{U \parr \uparrow P}$$

These new restrictions correspond to the needs we have when performing proof search, as the manipulation of negative pairs $\uparrow\downarrow$ is not required, while a positive pair $\downarrow\uparrow$ is needed to control the interaction between a negative and a positive. If a formula of the shape $U \oplus P$ is considered as the equivalent of a focused sequent $\vdash U, [P]$ then the use of the f rule is clear: it is the equivalent of the *decision* rule in the sequent calculus, where one positive is chosen to be treated. Then, the r rule correspond to the *reaction* rule, used when the interaction is completed — that is, when the subformula of a positive connective is a negative formula.

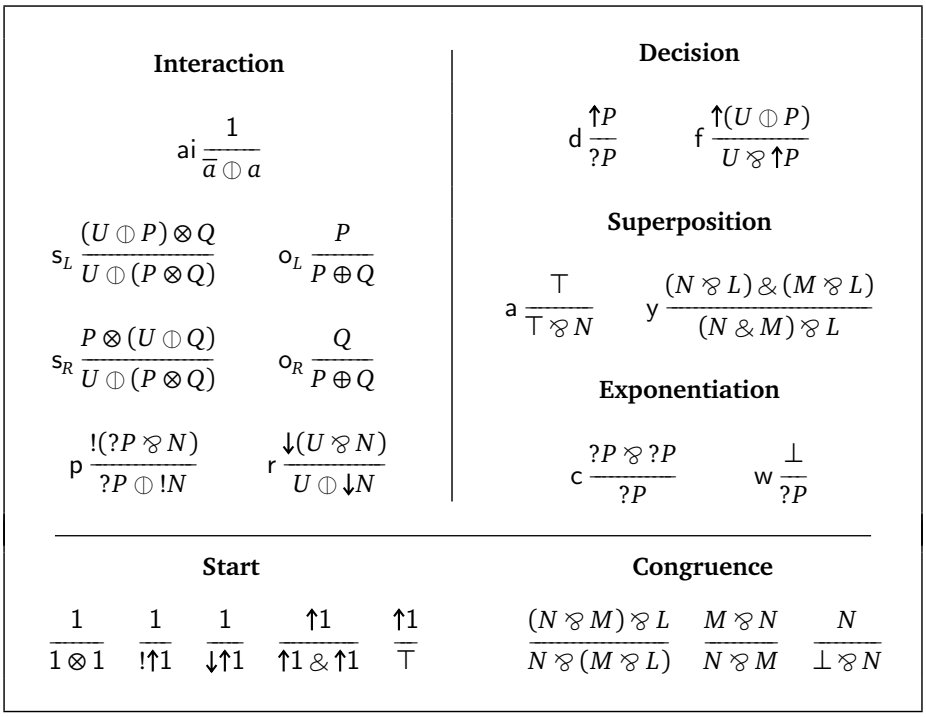


Figure 6: Inference rules for system LEF

In the design of the focused LEF system, we were guided by two ideas. First, the goal of making polarities explicit in the formulas and in inference rules has lead us to the LEP system. Then, the modifications of LEP leading to the LER system were motivated by the idea that shifts should not be introduced during proof search, or should in any case be manipulated only when strictly required. This goal was not achieved in LER because of the free use of the four rules for shift manipulation, but it is at least decoupled from other rules in this system. Finally, the LEF system is obtained by restricting the manipulations of shifts as much as possible, and because of the new connective $\oplus\!\!\!\!\cdot$ and its meaning in terms of polarities, this system never allows syntactical introduction of shifts during proof search — the shifts are hidden inside this connective.

It is interesting to notice that this reasoning, and the corresponding refinement of proof systems, leads to a system which is very similar[7] to the standard focused sequent calculus, as defined by Andreoli. Indeed, there are still more proofs in LEF than in the LLF focused sequent calculus, but this is normal in the setting of deep inference, and our goal was not to restrict the LSE system so much that it would be as weak as the sequent calculus, but the principles behind the design of LEF are the same as in the standard focusing setting.

[7] Of course, the system was developped with the standard focusing result in mind, but all the steps leading to it can be justified in terms of polarities, thus exposing their strong relation to focusing.

From a proof search perspective, the LEF system is meant to be used in a similar way as the LLF sequent calculus. We start with any given polarised formula, start treating negative formulas, and then perform one complete focusing phase. But the two parts of this » *dipole* « are slightly different than in the sequent setting, because they are decomposed variants of the asynchronous and synchronous phases. The asynchronous phase in LLF is meant to ensure that available negative connectives are decomposed before we pick a positive to decompose, since this positive might require to split the context obtained after the asynchronous phase. In LLF, only the negative formulas at toplevel in the sequent are available, but in LEF we might treat negative connectives deep inside the formula: the point is that only negative formulas in the » *direct context* « of a positive formula need to be treated when the positive involved in the subsequent focusing phase is this one. This direct context is the maximal context of the shape $\uparrow\xi$, where all the positive formulas inside ξ — including the considered positive P — are replaced with holes.

Once negative formulas in the surroundings of the formula P have been treated by the rules of the congruence, superposition and exponentiation fragments, this P can be associated to a reactive formula U, and prepared for an interaction. The preparation depends on how P appears in its negative context:

$$\mathsf{f}\,\frac{\uparrow(U \oplus P)}{U \parr \uparrow P} \qquad \text{or} \qquad \begin{array}{l}\mathsf{f}\,\dfrac{\uparrow(U \oplus P)}{U \parr \uparrow P}\\ \mathsf{d}\,\dfrac{}{U \parr ?P}\end{array}$$

where the congruence fragment is used to move U next to P. Finally, the interaction can be performed by the set of synchronous rules involving the $\oplus$ connective, and this corresponds to one *slice* of the synchronous decomposition of P as it would be performed in the sequent calculus. Then, the two steps are repeated until the proof is complete.

Just as in the sequent calculus, focusing is a *cyclic* normal form that repeats the basic operations of asynchronous and synchronous phases. By a detailed analysis of the possible permutations between rules of LEF, we could show that the following decomposition is possible:

$$\begin{array}{r} 1 \\ \textbf{Start} \parallel \\ Z \\ (\textbf{Focus})^* \parallel \\ A \end{array} \qquad \text{where } \textbf{Focus} \text{ is defined as} \qquad \begin{array}{r} A \\ \textbf{Interaction} \parallel \\ A_i \\ \textbf{Decision} \parallel \\ A_d \\ \textbf{Exponentiation} \parallel \\ A_e \\ \textbf{Superposition} \parallel \\ A_s \\ \textbf{Congruence} \parallel \\ A_c \end{array}$$

as it corresponds to the description of a focused strategy given above. The focusing result states that the LEF system is complete with respect to the unfocused system LSE, but we will prove this later and consider now a variant of LEF.

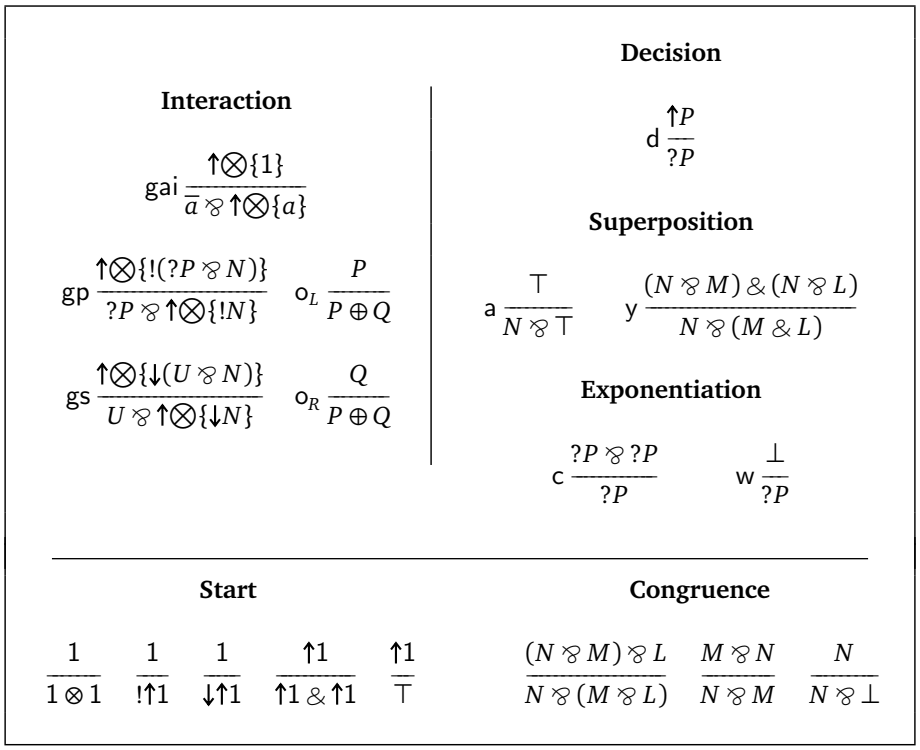


Figure 7: Inference rules for system LEG

3.2 The Grouped System LEG

While the LEF system allows the observation of the small steps leading a negative formula through a layer of positive connectives, one of the main consequences of using a focused system is that this can be hidden. Indeed, if we group rules in the interaction fragment, we can observe big steps, where a reactive formula is pushed through a complete positive layer at once. This is the idea behind the *synthetic rules* and synthetic connectives [Cha08] that stem from basic focused rules. Following this methodology, we introduce the *grouped*[8] system called LEG, where interaction between a reactive formula and a positive layer is done in big steps. It uses a special notation for layers of positive connectives.

Definition 3.9. *In the* LEG *system,* positive groups *are the outer-positive and positive contexts denoted by* $\bigotimes\{\ \}$ *or named* π *and defined by the following grammar:*

$$\pi\ ::=\ \{\ \}\ \mid\ \pi\otimes P\ \mid\ P\otimes\pi$$

The inference rules for LEG are shown in Figure 7, and one can notice how it is similar to LEF but hides the $\oplus$ connective through the use of synthetic rules.

[8] We choose not to call this system *synthetic* because it groups only the positive connectives together, while separated rules handle negative connectives individually, even if they could also be grouped.

The LEG system is a variant of the focused LEF system respecting the focusing discipline, by never introducing new shifts during proof search, but without relying on the interaction connective ⦶. To achieve this, it groups interaction rule instances into synthetic rule instances, so that only the *interface* of a focusing phase is seen, which never introduces shifts. This is an interesting variant of LEF because it uses the basic syntax of polarised formulas, and can be used without restrictions, but it ensures the really important part of focusing: synchronous steps are atomic.

Also, this system is an illustration of the dissymmetry of the focusing normal form with respect to the negative/positive categories. Rules for positive connectives are grouped, while rules for negative connectives can be applied separately, at any point in the proof, with the only constraint that a positive can only interact with a reactive formula, so that & formulas must duplicate the positive P before this P can start interacting with one of the conjuncts. This situation is radically different from the focused sequent calculus LLF, where the mandatory eager decomposition of negative formulas is an artifact of the shallow methodology. In the calculus of structures, this is not required because each reactive formula interacts separately with one positive, in the lazy way characteristic of the switch rule and of the nested setting in general. The important result here is completeness of LEG with respect to the unfocused LSE system, but we will also prove this in the next section. We can prove now that LEG is sound with respect to LEF.

Theorem 3.10. *If a polarised formula A is provable in the* LEG *system then it is also provable in the* LEF *focused system.*

Proof. This is straightforward, because only the rules gai, gp and gs are not also rules of LEF, and each instance of these rules can be replaced by a derivation of LEF — as can be shown by a simple induction on the structure of the positive group involved. □

4 Completeness and Relation to Sequent Calculi

The focused system LEF presented in the previous section, and its variant LEG, are the first sytems implementing the focusing methodology in another formalism than the sequent calculus — except for a focused system described in natural deduction [BNS10], but this is also a shallow setting — proving that it is not just an artifact of sequents, but rather an underlying principle of deductive reasoning. There are however two points to study in more details, the first and most important one being the proof of completeness of these focused calculi. In the calculus of structures, the proof of this result, which can be tedious in the sequent calculus, is surprisingly simple. Indeed, this boils down to the admissibility of one rule, which breaks the focusing property — just as the cut, which breaks the subformula property, can be shown admissible. Then, we can describe in more details the relation between the standard focused sequent calculus LLF and the focused calculi of structures proposed here. It turns out that there is a close correspondence between LLF and LEF, despite the radically different structure of their proofs.

4.1 Internal Proof of the Focusing Property

The focused proof system LEF, as well as its grouped variant LEG, could be proved complete with respect to linear logic by translation from the LL sequent calculus or its focused variant LLF. However, it is more interesting to consider an internal proof of the focusing result, independent from the sequent calculus, as it reveals more on the nature of the focusing normal form, and the corresponding transformation.

A remarkably simple way of proving completeness of the focusing restrictions in the calculus of structures is to consider the LEG system, extend it by an admissible rule which breaks the focusing property, and then show how the resulting system can simulate the unfocused calculus LSE. This elegant technique is similar to the traditional cut elimination result and supports the comparison between these two proof transformations, following the argument that respecting the polarity shifts of a formula is of the same nature as respecting the subformula property.

The rule we introduce in LEG to break the focusing property is called pg and performs *partial group crossing*, allowing any reactive formula to be moved inside a layer of positive connectives, but not necessarily at the exact negative border of this group, as would be done with the gs or gp rules:

$$\mathsf{pg}\ \frac{\uparrow\bigotimes\{\downarrow(U \parr \uparrow P)\}}{U \parr \uparrow\bigotimes\{P\}}$$

The resulting system does not respect the principle that polarity shifts should never be introduced in a formula during proof search, but there is no way of moving a reactive formula in the middle of a positive group without introducing a pair of shifts. This rule corresponds to the use of a delay $\downarrow\uparrow$, as expressed by the following lemma.

Lemma 4.1. *If there is a proof $\mathscr{P}$ of a formula $\xi\{\downarrow\uparrow P\}$ in* LEG*, then there is a proof of $\xi\{P\}$ in* LEG $\cup$ {pg} *of at most the same height as $\mathscr{P}$.*

Proof. We proceed by induction on the height of $\mathscr{P}$, with a base case when $\xi\{\downarrow\uparrow P\}$ is $\downarrow\uparrow 1$, where the result is immediate. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{P}$, and in most cases we can directly apply the induction hypothesis and remove the delay $\downarrow\uparrow$ from the formula. If r is an instance of the start rule rewriting 1 into $\downarrow\uparrow 1$, we can remove this instance and go on by induction hypothesis. Finally, if r is a matching gs instance, we replace it with an instance of pg, as follows:

$$\mathsf{gs}\ \frac{\xi\{\uparrow\bigotimes\{\downarrow(U \parr \uparrow P)\}\}}{\xi\{U \parr \uparrow\bigotimes\{\downarrow\uparrow P\}\}} \qquad \longrightarrow \qquad \mathsf{pg}\ \frac{\xi\{\uparrow\bigotimes\{\downarrow(U \parr \uparrow P)\}\}}{\xi\{U \parr \uparrow\bigotimes\{P\}\}}$$

□

However, the pg rule is admissible in the LEG system: this can be shown by a simple induction on the structure of a proof using it. The idea here is that when it is moved upwards, the pg rules is assimilated inside valid grouped rule instances, as one would » *repair* « a proof that does not respect polarities. The following lemma is the crucial step in the focusing proof.

Lemma 4.2. *The rule* pg *is admissible in* LEG.

Proof. Given a proof $\mathscr{P}$ of a formula A in $\mathsf{LEG} \cup \{\mathsf{pg}\}$, we prove by induction on the height of $\mathscr{P}$ that there is a proof of A in LEG of at most the same height. In the base case, A is 1 and the result is trivial. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{P}$, and if it is not an instance of pg, we use the induction hypothesis on the proof above r and compose the result with r. In the case where r is a pg instance, we consider the instance r_1 above r in $\mathscr{P}$ and use another case analysis:

1. If r_1 is not an instance of gai, gp, gs or pg then we can permute it above r in $\mathscr{P}$ and conclude by induction hypothesis on the proof above — notice that in the case of a c or y instance, we need to use it twice, which is possible because the transformation is height-preserving.

2. If r_1 is an instance of gai, we merge r and r_1 into a new gai instance:

$$\mathsf{gai}\,\dfrac{\mathsf{pg}\,\dfrac{\xi\{\uparrow\bigotimes_1\{\downarrow\uparrow\bigotimes_2\{1\}\}\}}{\xi\{\uparrow\bigotimes_1\{\downarrow(\overline{a}\parr\uparrow\bigotimes_2\{a\})\}\}}}{\xi\{\overline{a}\parr\uparrow\bigotimes_1\{\bigotimes_2\{a\}\}\}} \quad\longrightarrow\quad \mathsf{gai}\,\dfrac{\xi\{\uparrow\bigotimes_1\{\bigotimes_2\{1\}\}\}}{\xi\{\overline{a}\parr\uparrow\bigotimes_1\{\bigotimes_2\{a\}\}\}}$$

   then use the induction hypothesis on the proof above, and apply Lemma 4.1 on the result to produce a proof $\mathscr{P}'$ with a conclusion matching the premise of the new gai instance — and finally, we can use the induction hypothesis on $\mathscr{P}'$ and compose the result with the new instance.

3. If r_1 is an instance of gp, we merge r and r_1 into a new gp instance:

$$\mathsf{gp}\,\dfrac{\mathsf{pg}\,\dfrac{\xi\{\uparrow\bigotimes_1\{\downarrow\uparrow\bigotimes_2\{!(?P\parr N)\}\}\}}{\xi\{\uparrow\bigotimes_1\{\downarrow(?P\parr\uparrow\bigotimes_2\{!N\})\}\}}}{\xi\{?P\parr\uparrow\bigotimes_1\{\bigotimes_2\{!N\}\}\}} \quad\longrightarrow\quad \mathsf{gp}\,\dfrac{\xi\{\uparrow\bigotimes_1\{\bigotimes_2\{!(?P\parr N)\}\}\}}{\xi\{?P\parr\uparrow\bigotimes_1\{\bigotimes_2\{!N\}\}\}}$$

   and proceed the same way as in the previous case, using Lemma 4.1, and the induction hypothesis twice.

4. If r_1 is an instance of gs, we merge r and r_1 into a new gs instance:

$$\mathsf{gs}\,\dfrac{\mathsf{pg}\,\dfrac{\xi\{\uparrow\bigotimes_1\{\downarrow\uparrow\bigotimes_2\{\downarrow(U\parr N)\}\}\}}{\xi\{\uparrow\bigotimes_1\{\downarrow(U\parr\uparrow\bigotimes_2\{\downarrow N\})\}\}}}{\xi\{U\parr\uparrow\bigotimes_1\{\bigotimes_2\{\downarrow N\}\}\}} \quad\longrightarrow\quad \mathsf{gs}\,\dfrac{\xi\{\uparrow\bigotimes_1\{\bigotimes_2\{\downarrow(U\parr N)\}\}\}}{\xi\{U\parr\uparrow\bigotimes_1\{\bigotimes_2\{\downarrow N\}\}\}}$$

   and proceed the same way as in the previous case, using Lemma 4.1, and the induction hypothesis twice.

5. If r_1 is also a pg instance, then we use the induction hypothesis on the proof above r to obtain a proof $\mathscr{P}'$ on which we use again the induction hypothesis.

In the end of the process, we obtain a proof where all the pg instances have been merged into instances of other grouped rules, therefore valid in LEG. □

The second step is to show that the pg rule, by its ability to break the focusing discipline, allows to simulate any given proof from an unfocused system. The basis used here to write unfocused proofs is LER, since it is the closest unfocused system that we have defined, where some restrictions of the focused system have already been shown complete.

Lemma 4.3. *A polarised formula A is provable in* LER *if and only if it is provable in the grouped system* LEG ∪ {pg} *with partial group crossing.*

Proof. Given a proof of A in LEG ∪ {pg}, it is trivial to build a proof of A in LER, since any rule instance in LEG ∪ {pg} can be replaced with a derivation in LER— in the case of the grouped rules gai, gp and gs, this can be shown by a straightforward induction on the structure of the positive group involved.

In the other direction, we can prove by induction on the height of a given proof $\mathscr{P}$ of A in LER how to build a proof of A in LEG ∪ {pg}. In the base case, A is 1 and the result is trivial. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{P}$, and there are only four interesting cases, the ones of the rules ai, s_L and s_R, and p, which are actually particular cases of the rules gai, pg, and gp respectively:

$$\mathsf{gai}\,\frac{\uparrow 1}{\overline{a} \parr \uparrow a} \qquad \mathsf{pg}\,\frac{\uparrow(\downarrow(U \parr \uparrow P) \otimes Q)}{U \parr \uparrow(P \otimes Q)} \qquad \mathsf{pg}\,\frac{\uparrow(P \otimes \downarrow(U \parr \uparrow P))}{U \parr \uparrow(P \otimes Q)} \qquad \mathsf{gp}\,\frac{\uparrow!(?P \parr N)}{?P \parr \uparrow!N}$$

and we can conclude by induction hypothesis, after these replacements. In all other cases, we can simply reuse r and conclude by induction hypothesis, since the other rules of LER are valid in LEG ∪ {pg} too. □

This proves that pg is really the missing piece between the unfocused and the focused systems. Now, the actual focusing result is obtained by assembling the lemmas to create a connection between the basic LSE system and the focused LEF system, using the fact that given a proof in LEG, we can turn it into a proof in LEF.

Theorem 4.4 (Focusing)**.** *Any polarised formula A is provable in the focused system* LEF *if its unpolarised variant $\lceil A \rceil$ is provable in the unfocused system* LSE.

Proof. Given a proof of $\lceil A \rceil$ in LSE, we can apply Theorem 2.9 to translate it to a proof of A in the polarised system LEP. Then, by applying Lemma 3.8 we can build build a proof in LER and subsequently in LEG ∪ {pg}, using Lemma 4.3. Thus by Lemma 4.2 we obtain a proof of A in LEG. Finally, by Theorem 3.10 we have the expected proof of A in LEF. □

This internal completeness result can be obtained directly in the LEF system, without going through the grouped variant LEG, by proving the admissibility of the rule shown below, which adapts the idea of pg to the interaction connective of the LEF system:

$$\frac{\downarrow(U \parr \uparrow P)}{U \oplus P}$$

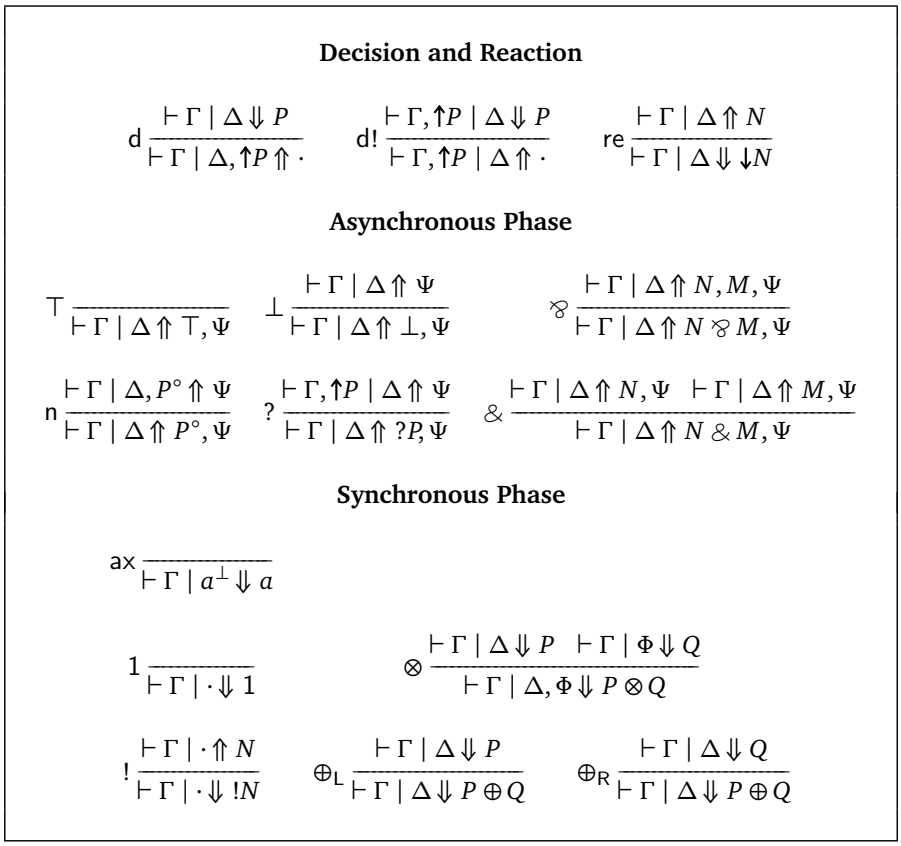


Figure 8: Inference rules for the focused system LLF

4.2 Correspondence to Standard Focusing

Since the focusing technique was introduced in the setting of the sequent calculus for linear logic, in which it has been thoroughly studied, it is interesting to compare the so-called focused calculus of structures that we have proposed in the previous section. The LLF sequent calculus, presented in Chapter 2, is recalled in Figure 8, where the standard triadic syntax is used, but it is slightly modified to use polarised formulas rather than plain linear logic formulas. In this setting, P° denotes either $\uparrow P$ or a negative atom $\overline{a}$.

From LLF to LEF. The first part of the comparison consists in the simulation of focused sequent proofs in the LEF calculus of structures. The difficulty there lies in the difference between shallow and nested formalisms, but we can establish a correspondence through an abstraction relation that relates formulas in LEF to the triadic sequents of the LLF calculus. The rest of the reasoning will be done *modulo* this abstraction.

Definition 4.5. *Given a triadic sequent δ and a polarised formula A, we say that A is a* structural interpretation *of δ, written $A \approx \delta$, if we can derive it from the rules:*

$$\frac{}{P \approx (\vdash \cdot \mid \cdot \Downarrow P)} \qquad \frac{P \approx (\vdash \Gamma \mid \Delta \Downarrow Q)}{(U \oplus P) \approx (\vdash \Gamma \mid \Delta, U \Downarrow Q)} \qquad \frac{N \approx (\vdash \Gamma \mid \Delta \Uparrow \Psi)}{(N \parr M) \approx (\vdash \Gamma \mid \Delta \Uparrow M, \Psi)}$$

$$\frac{}{N \approx (\vdash \cdot \mid \cdot \Uparrow N)} \qquad \frac{P \approx (\vdash \Gamma \mid \Delta \Downarrow Q)}{(?R \oplus P) \approx (\vdash \Gamma, \uparrow R \mid \Delta \Downarrow Q)} \qquad \frac{N \approx (\vdash \Gamma, \uparrow R \mid \Delta \Uparrow \Psi)}{(N \parr ?R) \approx (\vdash \Gamma, \uparrow R \mid \Delta \Uparrow \Psi)}$$

$$\frac{P \approx (\vdash \Gamma \mid \Delta \Downarrow Q)}{P \approx (\vdash \Gamma, \uparrow R \mid \Delta \Downarrow Q)} \qquad \frac{N \approx (\vdash \Gamma \mid \Delta \Uparrow \Psi)}{N \approx (\vdash \Gamma, \uparrow R \mid \Delta \Uparrow \Psi)}$$

With this definition, structural interpretations can arbitrarily reorder a sequent, but they preserve the multiplicities of the formulas in the linear part of the LLF sequent, while potentially erasing or duplicating the unrestricted formulas. Then, we can prove a simulation theorem showing that LEF can preserve the structural interpretations of each rule of LLF.

Remark 4.6. *In the following, we will use the notations $\Delta \oplus P$ and $\Delta \parr N$ respectively for $M_1 \oplus (\cdots \oplus (M_n \oplus P))$ and $M_1 \parr \cdots \parr M_n \parr N$ to simplify the handling of the multisets of the form $\Delta = M_1, \cdots, M_n$ from the sequent calculus, with the convention that this is simply P and N respectively in the case where Δ is empty.*

Theorem 4.7. *For any Γ, Δ and P, if $\vdash \Gamma \mid \Delta \Downarrow P$ is provable in* LLF *then there is a $Q \approx (\vdash \Gamma \mid \Delta \Downarrow P)$ such that Q is provable in* LEF*, and for any Ψ, if $\vdash \Gamma \mid \Delta \Uparrow \Psi$ is provable in* LLF *then there is a $N \approx (\vdash \Gamma \mid \Delta \Uparrow \Psi)$ provable in* LEF*.*

Proof. Given a proof $\mathscr{P}$ in LLF, we proceed by structural induction on $\mathscr{P}$, with a trivial base case when $\mathscr{P}$ is reduced to an identity instance, so that we can use the ai rule from LEF:

$$\mathsf{ax}\,\frac{}{\vdash \Gamma \mid a^\perp \Downarrow a} \qquad \longrightarrow \qquad \mathsf{ai}\,\frac{1}{\overline{a} \oplus a}$$

In the general configuration, we consider all the cases for the bottommost rule instance in $\mathscr{P}$ to perform the structural induction. All cases are similar, and we show the case of the $\otimes$ rule:

$$\otimes\,\frac{\vdash \Gamma \mid \Delta \Downarrow P \quad \vdash \Gamma \mid \Phi \Downarrow Q}{\vdash \Gamma \mid \Delta, \Phi \Downarrow P \otimes Q} \qquad \longrightarrow \qquad \begin{array}{c} \dfrac{1}{1 \otimes 1} \\ \mathscr{P}_2 \| \\ 1 \otimes (\Sigma_2 \oplus Q) \\ \mathscr{P}_1 \| \\ (\Sigma_1 \oplus P) \otimes (\Sigma_2 \oplus Q) \\ \{\mathsf{s}_L, \mathsf{s}_R\} \| \\ \Sigma_1 \oplus (P \otimes (\Sigma_2 \oplus Q)) \\ \{\mathsf{s}_L, \mathsf{s}_R\} \| \\ \Sigma_1, \Sigma_2 \oplus (P \otimes Q) \end{array}$$

where $(\Sigma_1 \oplus P) \approx (\vdash \Gamma \mid \Delta \Downarrow P)$ and $(\Sigma_2 \oplus Q) \approx (\vdash \Gamma \mid \Psi \Downarrow Q)$, so that the proofs $\mathscr{P}_1$ and $\mathscr{P}_2$ are obtained by induction hypothesis. □

Corollary 4.8 (Completeness)**.** *If the sequent* $\vdash \Gamma \mid \Delta \Uparrow \Psi$ *is provable in* LLF*, then the structure* $\Delta \parr \Psi \parr ?\Gamma$ *is provable in* LEF*.*

Proof. Given the structure $\Delta \parr \Psi \parr ?\Gamma$, we can apply the rules c and w to build a derivation with $\Delta \parr \Psi \parr ?\Sigma$ as premise, where Σ is Γ where some structures $?P$ have been erased or duplicated. Then, we can apply Theorem 4.7 to conclude. □

This result allows us to represent any LLF proof in the LEF system in a shallow form, where the interactions happen only at the outermost level. This is not using much of the particular features of LEF, but provides a translation from the focused sequent calculus into the focused calculus of structures.

From LEF **to** LLF. The other way around, we might want to consider the LEF proofs where the interactions happen anywhere inside structures, and we can show that a single LLF proof — *modulo* some irrelevant permutations of negative rules — can be extracted from any LEF proof.

Given a proof $\mathscr{P}$ of some structure $\downarrow N$, we can decompose $\mathscr{P}$ into a derivation from $\downarrow M$ to $\downarrow N$ in $\{\mathsf{c}\}$, for some M, and a proof $\mathscr{P}'$ of $\downarrow M$ in $\mathsf{LEF} \setminus \{\mathsf{c}\}$, as can be shown by a straightforward inductive argument. Indeed, it is easy to observe that contraction permutes below all other inference rules of the LEF system. Then, we extract information from the proof $\mathscr{P}'$ by uniquely labelling the active and reactive formulas in M. More precisely, we modify the grammar of structures as follows:

$$\begin{aligned} P,Q &::= a_u \mid 1 \mid P \otimes Q \mid 0 \mid P \oplus Q \mid !_u N \mid \downarrow_u N \mid N \invamp P \\ N,M &::= \overline{a}_u \mid \bot \mid N \parr M \mid \top \mid N \with M \mid ?_u P \mid \uparrow_u P \end{aligned}$$

where letters such as u or v denote labels drawn from an infinite set. Then, the rules for LEF are modified to incorporate these labels. Most cases are straightforward, and the important rules are the following:

$$\mathsf{f}\,\frac{\uparrow_u (U_v \invamp P)}{U_v \parr \uparrow_u P} \qquad \mathsf{ai}\,\frac{}{\overline{a}_v \invamp a_u}\,1 \qquad \mathsf{p}\,\frac{!_u(?_v P \parr N)}{?_v P \invamp !_u N} \qquad \mathsf{r}\,\frac{\downarrow_u (U_v \parr N)}{U_v \invamp \downarrow_u N}$$

$$\mathsf{d}\,\frac{\uparrow_u P}{?_u P} \qquad \frac{\uparrow_u 1}{\uparrow_u 1 \with \uparrow_v 1} \qquad \mathsf{c}\,\frac{?P_u \parr ?_v P}{?_u P}$$

where the two labels u and v in the c rule are said to be of the same *species* — this is needed to keep track of the relation between structures obtained by duplication of the same initial structure. The rules $\{\mathsf{f}, \mathsf{ai}, \mathsf{p}, \mathsf{r}\}$ in the first line induce an ordering $<$ on labels, with $u < v$ in each case. Since we have labelled the conclusion of $\mathscr{P}'$ in such a way that all active and reactive formulas have unique labels, the reflexive, transitive closure of this label ordering is a partial order denoted by $\leq$.

We will extract a proof of $\vdash \cdot \mid \cdot \Uparrow N$ in LLF by an algorithm using this order. The algorithm is essentially deterministic, since the only choices it makes are the order in which it applies negative rules — there is no choice involved in the positive or decision rules. We use a labelled version of LLF where the unrestricted context contains reactive formulas labelled with a multiset of labels of the same species, so that they are written $\uparrow_L P$, where $L = u_1, \cdots, u_n$.

Then, the non-linear decision rule d! is modified to consume one of the available labels in the multiset of the formula, and the d rule also consumes the label of the considered linear reactive formula. Moreover, the exponential rule ? is modified to extend one of the label multisets corresponding to the same species of label — if there is no existing formula in the conclusion of the instance with the same species of label, the rule is interpreted as implicitly adding one. The resulting rules are:

$$\mathsf{d}\,\frac{\vdash \Gamma \mid \Delta \Downarrow P}{\vdash \Gamma \mid \Delta, \uparrow_u P \Uparrow \cdot} \qquad \mathsf{d!}\,\frac{\vdash \Gamma, \uparrow_L P \mid \Delta \Downarrow P}{\vdash \Gamma, \uparrow_{L,u} P \mid \Delta \Uparrow \cdot} \qquad ?\,\frac{\vdash \Gamma, \uparrow_{L,u} P \mid \Delta \Uparrow \Psi}{\vdash \Gamma, \uparrow P_L \mid \Delta \Uparrow ?_u P, \Psi}$$

and the remaining rules of LLF are modified to use labels in a straightforward way. The extraction algorithm for a labelled LLF proof of $\vdash \cdot \mid \cdot \Uparrow N$ then proceeds by applying LLF rules upwards, with the following cases:

1. For a sequent $\vdash \Gamma \mid \Delta \Uparrow \Psi$ for which there are available negative rules to be applied, one of these rules is applied, and the order of the application of the rules is irrelevant.

2. For a sequent $\vdash \Gamma \mid \Delta \Uparrow \cdot$ we pick the unique, smallest label for $\leq$ from the available labels in Γ and Δ, and use d or d! accordingly — the completeness of the whole algorithm depends on the presence of such a smallest label.

3. For the $\otimes$ rule of LLF, we send side formulas to the premise involving P when their labels are larger by $\leq$ than some label of a subformula of P, and the rest to the premise involving Q — since labelling is unique, the sets of labels in P and Q are disjoint —, and for the $\oplus$ rules we simply repeat the choice made in the $\mathscr{P}'$ proof.

Notice that there are no labels in 1, so that nothing can be sent to any branch focusing on 1, and the only side formula that can be sent to a branch focusing on a_u is a formula $\overline{a}_v$ with $u < v$, so that the identity rule always succeeds. This algorithm leads us to the expected result.

Proposition 4.9 (Extraction)**.** *If a structure* $\downarrow N$ *is provable in* LEF *then the sequent* $\vdash \cdot \mid \cdot \Uparrow N$ *is provable in* LLF.

This relies on the fact that at any step of the bounded search described above, there exists a unique label smallest for $\leq$ among all available labels. This implies that the algorithm can always make progress, and it is also terminating because labels are consumed by the d and d! rules. From $\downarrow N$ we can thus build a proof in LLF starting with a reaction instance, that we can cut off, and we finally remove all labels to obtain the desired proof.

Chapter 8

Proof Search as Reduction in the λ-calculus

In this chapter, we present a non-standard correspondence between proof search in an intuitionistic system and computation as described in the λ-calculus. Due to the collapse of the branches into a flat sequential structure, a proof in the calculus of structures is built by a sequence of rewriting steps on formulas, opening the way for an interpretation of proof search as reduction in a computational model based on rewriting. Naturally, the λ-calculus is an interesting candidate, and we set out to relate proof search in the JS proof system, described in Chapter 4, to the reduction system of a λ-calculus with explicit substitutions.

However, the proof search process in a nested deduction system is much richer than the dynamics of explicit substitutions, and we need to impose restrictions on the logical system to make it match the chosen computational model. There is an obvious mismatch in the sense that the result of a computation is not always the same, as the premise of a proof is always the truth unit, but this simply means that we will be concerned here with open derivations rather than complete proofs. The most important restrictions are those ensuring an adequate level of determinacy in the application of inference rules, and of course the ones necessary to make the structure of formulas match the structure of a λ-term with explicit substitutions.

We start with the plain JS system and define several restrictions to obtain our candidate logical system. Then, we show that all the restrictions on inference rules are reasonable with respect to the restrictions made on the set of formulas, and we exhibit a direct, one-to-one correspondence between inference rules of this system and reduction rules for the $\lambda\mathtt{s}$-calculus, presented in Chapter 2, through a simple encoding. We then discuss how this strong relation between logic and computation allows to transfer results from one world to the other.

Finally, we observe that through this correspondence, the strategy chosen to perform proof search on a formula is also the strategy chosen to reduce the term it represents. Following this idea, we introduce further restrictions on the side of the logical system, using annotations to enforce a normal form in the style of focusing, and show how it creates phases corresponding to big-step reductions implementing the call-by-name reduction strategy of the standard λ-calculus.

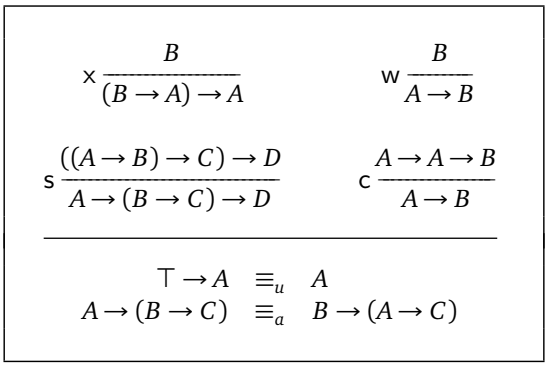


Figure 1: Inference rules and congruence for JS

1 Proof Search as Rewriting

In the calculus of structures, the notion of deduction is not implemented through rules decomposing formulas into a deductive meta-level, as in the sequent calculus, but rather through rewriting rules that can be applied anywhere in formulas. This approach to the representation of proof objects establishes a connection between the standard framework of proof theory and the well-developped theory of rewrite systems [BN98], that is not just the traditional view of proof normalisation, or cut elimination, as a rewriting operation on the term representing a proof.

The operations performed on the structures of a system such as JS, presented in Chapter 4, and recalled in Figure 1, are much more complex than the manipulation of formulas in the sequent calculus or natural deduction. It is actually rich enough to be rather similar to some rewrite systems found in the literature: in particular, we are interested here in its similarity to the λ-calculus with explicit substitutions, as presented in Chapter 2, in its standard version.

The connection between the λ-calculus and proof search in the setting of deep inference has already been studied in the particular, restricted case of the purely linear λ-calculus [Rov11]. However, this correspondence relies on a rather exotic proof system, which is a variant of the system BV [Gug07] extended by an operator for renaming atomic formulas. Moreover, the conceptual complexity of the system is in mismatch with the simplicity of the linear λ-calculus, and extending this to the standard λ-calculus would be complicated because of the complexity of exponentials in linear logic — a system such as NEL [GS02] would be required.

We propose a study of the JS intuitionistic system based on three observations:

(i) Proof search in deep inference allows the application of an inference rule at any position inside a formula.

(ii) Implication in the intuitionistic system JS *is a non-commutative connective that induces a distinction between positive and negative positions.*

(iii) All inference rules in JS *preserve the positive and negative position of formulas, even when formulas are moved from one context to another.*

The first observation is obviously the reason why encoding any computational model based on rewriting, such as the λ-calculus, is possible in the first place. The two other observations are suggesting that it is not necessary to introduce a new non-commutative connective as in BV, since implication is non-commutative and has the required properties — in particular, it provides a clear distinction between positive and negative formulas.

Following the idea that we have two separate kinds of formulas in JS, we decide to represent λ-terms as positive formulas and binding names as negative ones. We want to establish a correspondence based on the following encoding of a λ-term into an intuitionistic structure:

$$\begin{array}{rcl} [\![x]\!] & = & x \\ [\![\lambda x.t]\!] & = & x \to [\![t]\!] \\ [\![t\ u]\!] & = & ([\![u]\!] \to \top) \to [\![t]\!] \\ [\![t[x \leftarrow u]]\!] & = & ([\![u]\!] \to x) \to [\![t]\!] \end{array}$$

which reflects the idea that an application, that can become a β-redex, is a potential explicit substitution to which no binding name has yet been attached. In order to support the correspondence, we will need a proof system where the application of an inference rule rewrites the encoding of a term t into the encoding of a term u such that $t \longrightarrow u$. This requires the use of explicit substitutions, through the use of the $\lambda\mathtt{s}$-calculus [KR11], since the rewriting steps implemented in JS are primitive, local operations. The restricted shape of structures obtained through the encoding of terms leads to the use of a restriction of JS, called JSL, that ensures the stability of the encoding under inference.

Moreover, we will consider a variant of the JSL proof system using syntactic annotations in the style of focusing, which allows to consider groups of inference rule instances corresponding to the dispatch of the explicit substitutions inside all subterms of a term. This system yields a correspondence with big-step reduction of $\lambda\mathtt{s}$-terms, and therefore with the standard λ-calculus with plain β-reduction. On the other side of the restriction spectrum the question of an interpretation of proof search in the complete, unrestricted JS system remains open. The general form of the JS structures allowing complex negative substructures suggests an extension of binding names into *patterns*, yielding a correspondence with some calculus based on pattern-matching [JK09]. The general picture of the situation is:

Logical system	**Computational device**
JSL with restrictions and deterministic proof search	$\lambda\mathtt{s}$-calculus (explicit substitutions)
JSL with restrictions and focusing annotations	pure λ-calculus (with β-reduction)
JSL without further restrictions	non-deterministic variant of the λ-calculus ?
complete JS system	» *pattern calculus* « ?

2 A Restricted Intuitionistic System

Although it is a simple system for a fragment of intuitionistic logic, JS is already too general to represent the λ-calculus with precision in a proof search style. Indeed, no λ-term corresponds to a formula with a compound formula in negative position, such as $(A \to (B \to C)) \to D$, and even if we restrict formulas to images of the translation mentioned in the previous section, we still have problems with inference rules and equations that do not correspond to valid operations of our λ-calculus, as for example:

$$\begin{array}{rcl} \lambda x.t & \longrightarrow & \lambda x.\lambda x.t \\ \lambda x.\lambda y.t & \equiv_a & \lambda y.\lambda x.t \end{array}$$

Thus, we define a restriction of this system, called JSL, where formulas are limited to those where the subformulas in negative position, located on the left of an odd number of implications, only have a certain shape where at most one implication is used — plus some restrictions on the use of the $\top$ unit.

Definition 2.1. *The* restricted *formulas of intuitionistic logic are defined by B in the following grammar:*

$$B ::= a \mid a \to B \mid \omega \to B \mid \delta \to B$$

where a block δ *and a* matcher ω *are defined as:*

$$\delta ::= B \to a \qquad\qquad \omega ::= B \to \top$$

We will denote restricted formulas the same way as formulas, to keep notations simple. Blocks are defined as subformulas in negative position where the rightmost literal is an atom, they are denoted by greek letters such as δ, κ or μ, and they represent explicit substitutions in our encoding. Then, matchers are subformulas in negative position where the rightmost literal is the unit $\top$, they are denoted by ω or τ, and they represent the argument term in an application. We use contexts to restrict the congruence as well, keeping only the equation $\equiv_a$ on specific positive subformulas:

$$\xi\{\delta \to (\kappa \to A)\} \equiv_b \xi\{\kappa \to (\delta \to A)\}$$

so that only negative subformulas can be exchanged, and this corresponds to the fact that there is only one positive formula located directly under an implication in negative position, in the definition of restricted formulas. Notice that the equation $\equiv_b$ forbids the exchange of any subformula that is not a block.

Definition 2.2. *The* restricted *structures of the intuitionistic system* JSL *are defined as the equivalence classes of formulas generated by the congruence described by the single equation* $\equiv_b$.

Finally, the inference rules for the restricted system JSL are given in Figure 2. In this system, the switch rule and the rules of weakening and contraction can only be used to move, erase or duplicate blocks. Moreover, another part of the equation $\equiv_a$ is expressed in the rule ex, which also can only exchange blocks. The identity rule is also restricted, through the condition that the premise cannot be the $\top$ unit, and it only applies on atoms. Note that in the sr rule, L denotes a literal, which is either an atom a or the unit $\top$.

$$\mathsf{xr}\ \frac{B}{(B \to a) \to a} \qquad \mathsf{xsu}\ \frac{(B \to a) \to C}{(B \to \top) \to (a \to C)}$$

$$\mathsf{wr}\ \frac{A}{\delta \to A} \qquad \mathsf{cr}\ \frac{\delta \to (\delta \to A)}{\delta \to A}$$

$$\mathsf{ex}\ \frac{A \to (\delta \to B)}{\delta \to (A \to B)} \qquad \mathsf{sr}\ \frac{((\delta \to A) \to L) \to B}{\delta \to ((A \to L) \to B)}$$

Figure 2: Inference rules for system JSL

2.1 Restrictions on Structures and Rules

The new rule xsu is a compound rule that embodies the use of an identity on the unit $\top$, and corresponds to the following JS derivation:

$$\equiv_{\mathsf{u}} \frac{\mathsf{s}\ \dfrac{\mathsf{x}\ \dfrac{(B \to a) \to C}{((((B \to \top) \to \top) \to a) \to C)}}{(B \to \top) \to ((\top \to a) \to C)}}{(B \to \top) \to (a \to C)}$$

This new inference rule will correspond, through the encoding of $\lambda\mathtt{s}$-terms, to the standard B rule which triggers a β-redex by turning an application into an explicit substitution, to be carried out:

$$(\lambda x.t)\, u \longrightarrow_{\mathtt{B}} t[x \leftarrow u]$$

This system is clearly not going to be complete, since restrictions are so strong that for example, there is no way of writing a proof of $a \to (a \to \top) \to a$. However, this system can easily be shown sound with respect to its general version JS. This is proved in a strong sense, relating derivations of the two systems rather than just proofs. Moreover, it does not require any translation, since any restricted structure of JSL is a valid structure of JS[1].

Theorem 2.3 (Soundness of JSL)**.** *If there is a derivation from A to B in* JSL*, then there is a derivation from A to B in* JS*.*

Proof. Any derivation from A to B in JSL can be immediately converted into a derivation in JS. Indeed, the rules xr, wr, cr and sr are restrictions of the rules of JS, the equation $\equiv_b$ is a restriction of $\equiv_a$ and the ex rule also corresponds to another use of this equation. Finally, any instance of the xsu rule can be replaced with a derivation of JS, as shown above. □

[1]The congruence used in JSL is also a subset of the general congruence we used in JS, so that the equivalence class of a restricted formula, forming a structure, is larger in JS than in JSL.

It is interesting to notice that the identity on the unit $\top$ we use in the derivation corresponding to xsu is not even needed in JS, but will be shown admissible. To preserve the upper bound on the height of proofs during the process, it is useful to consider the system JS', a variant of JS where the usual switch is replaced with a compound rule called *super-switch* [Str03a]:

$$\mathsf{ss}\,\frac{\xi\{\delta\} \to B}{\delta \to (\xi\{\top\} \to B)}$$

which is equivalent to a derivation of several switches, so that there exist obvious translations between JS and JS'. We use this alternative system to count a sequence of switches as only one rule instance in the height of a proof.

Remark 2.4. *The* xsu *rule is not admissible in the restricted* JSL *system, since that would require an equation allowing to add a unit on the left of a formula, under an implication. This is clearly valid in intuitionistic logic but does not fit our restrictions.*

Proposition 2.5. *If there is a proof of a structure* $\xi\{A\}$ *in* JS*, then there is a proof of* $\xi\{(A \to \top) \to \top\}$ *as well in* JS*, not using an identity on these* $\top$ *occurrences.*

Proof. By induction on the height of the given proof $\mathscr{D}$ of A in JS, translated into the JS' system. If $\mathscr{D}$ is of height 0, we replace $\top$ with $(\top \to \top) \to \top$ using $\equiv_u$. Then, in the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$. We can always rewrite the conclusion and use the induction hypothesis to rewrite the premise, but in the case of a switch moving some structure E inside A, where A is $(B \to C) \to D$, we must use an additional switch:

$$\mathsf{s}\,\frac{\xi\{((E \to B) \to C) \to D\}}{\xi\{E \to ((B \to C) \to D)\}}$$

$$\longrightarrow \quad \mathsf{s}\,\frac{\mathsf{s}\,\dfrac{\xi\{((((E \to B) \to C) \to D) \to \top) \to \top\}}{\xi\{((E \to ((B \to C) \to D)) \to \top) \to \top\}}}{\xi\{E \to ((((B \to C) \to D) \to \top) \to \top)\}}$$

which can be rewritten into a super-switch ss instance. Note that in the case of the contraction c, we need to use the induction hypothesis twice, and this is possible because the transformation is height-preserving, thanks to the super-switch. □

Now, we will show that the JSL proof system is actually not so far from being complete with respect to the restricted fragment of intuitionistic logic. In order to do this, we will consider a smaller fragment, by imposing even more restrictions on structures: only one negative occurrence of each atom is allowed, and all of its positive occurrences must appear *» in the scope «* of this negative occurrence. We use the notation $a \in B$ to denote that some atom a appears in the structure B, and $a \in \xi$ if a appears in the context $\xi\{\ \}$.

Definition 2.6. *The* (positive) multiplicity *of an atom* a *in some structure* B*, denoted by* $|B|_a^+$*, is the number of occurrences of* a *in positive position within* B*. Its* negative multiplicity*, denoted by* $|B|_a^-$*, is its number of occurrences in negative position in* B*.*

$$
\begin{array}{ll}
\textit{apply } \mathsf{wr} \textit{ on } (B \to a) \to C & : \textit{ only if } |C|_a^+ = 0 \\
\textit{apply } \mathsf{crr} \textit{ on } (B \to a) \to C & : \textit{ only if } |C|_a^+ \geq 2 \\
\textit{apply } \mathsf{ex} \textit{ on } (B \to a) \to (C \to D) & : \textit{ only if } |C|_a^+ = 0, |D|_a^+ \geq 1 \\
\textit{apply } \mathsf{sr} \textit{ on } (B \to a) \to ((C \to L) \to D) & : \textit{ only if } |C|_a^+ \geq 1, |D|_a^+ = 0
\end{array}
$$

$$\xi\{(C \to a) \to ((D \to b) \to E)\} \equiv_b \xi\{(D \to b) \to ((C \to a) \to E)\}$$

this equation holds only if we have $|D|_a^+ = 0$ *and* $|C|_b^+ = 0$

Figure 3: Restrictions used to define JSLd from JSL

The formulas left in the more restricted class that we define are called *functional structures* because they will be exactly the formulas corresponding to $\lambda\mathfrak{s}$-terms in the following section.

Definition 2.7. *A restricted structure B is said to be* functional *if for any $a \in B$, there is a context $\xi\{\ \}$ and structures C and D such that we have either $B \equiv_b \xi\{a \to C\}$ or $B \equiv_b \xi\{(D \to a) \to C\}$, with $a \notin \xi$, $a \notin D$, $|C|_a^+ \geq 0$ and $|C|_a^- = 0$.*

This means that in any functional structure, a given atom can appear only once in negative position, but possibly many times in positive position. Then, we observe that functional structures are not stable under application of inference rules of JSL, in particular under contraction. Therefore we need to tweak our inference rules. To be able to use contraction on such structures, we consider the following variation of the cr rule:

$$\mathsf{crr}\ \frac{(B \to d) \to ((B \to a) \to C_{[d/a]})}{(B \to a) \to C}$$

where $C_{[d/a]}$ denotes[2] C with exactly one occurrence of the atom a replaced by an occurrence of a » *fresh* « atom d — not used in the rest of the structure. This rule can be shown sound through a straightforward induction on proofs. Finally, we define the proof system JSLd as $\{\mathsf{xr}, \mathsf{xsu}, \mathsf{wr}, \mathsf{crr}, \mathsf{ex}, \mathsf{sr}\}$, with extra conditions on the conclusion of rules and on the congruence, summarised in Figure 3. These conditions ensure that functional structures are stable under application of rules of JSLd, as well as its congruence.

These new restrictions correspond, on the side of our λ-calculus, to the idea that we want to manipulate terms up to renaming of variables, so that no variable name is bound twice, and we can use implicitly α-conversion to change names.

The JSLd system is sound with respect to intuitionistic logic, since we have only added restrictions on inference rules of JSL and we observed that the variant rule crr was sound too, and we now study the question of completeness. As we already noticed, this system is not complete — the unit $\top$ cannot appear as the premise of a derivation, so that there is no proof *per se* in this system — but we can establish

[2] This notation is reminiscent of the renaming operator that can be used in the λ-calculus with explicit substitutions [AK10], to handle duplication.

a partial completeness result. We do that in three steps: first we use a subset of the JSLd system, then we deal with some particular weakenings forbidden in JSLd, and finally we build the proof premise $\top$ from a simple formula, by dealing again with weakenings. The goal is to show that if some A is provable in JS, we can build a proof of the shape:

$$\begin{array}{r} \top \\ {\scriptstyle \{\mathsf{xw},\mathsf{uw}\}}\| \\ B \\ {\scriptstyle \mathsf{JSLb}}\| \\ A \end{array} \tag{11}$$

where JSLb is the subset of JSLd we will consider, and A is a functional structure, the idea being that our restricted rules are enough to prove this kind of restricted structures, up to particular weakenings.

2.2 Termination and the JSLb Proof System

We consider the system JSLb, which is defined as JSLd without the xsu rule, for which we show that proof search is terminating. To do that, we need a measure on functional structures that will decrease during proof search, and the first part of this measure can be defined in a simple way.

Definition 2.8. *Given a functional structure A, we define as follows its* block-complexity*, denoted by $\mathtt{C}(A)$, and its* net size*, denoted by $\mathtt{N}(A)$, using the following induction:*

$$\begin{array}{rcl} \mathtt{C}(a) & = & 0 \\ \mathtt{C}(a \to B) & = & \mathtt{C}(B) \\ \mathtt{C}((C \to \top) \to B) & = & \mathtt{C}(C) + \mathtt{C}(B) \\ \mathtt{C}((C \to a) \to B) & = & \mathtt{C}(C) + \mathtt{C}(B) + \mathtt{N}(B) \end{array}$$

$$\begin{array}{rcl} \mathtt{N}(a) & = & 1 \\ \mathtt{N}(a \to B) & = & 1 + \mathtt{N}(B) \\ \mathtt{N}((C \to \top) \to B) & = & \mathtt{N}(C) + \mathtt{N}(B) \\ \mathtt{N}((C \to a) \to B) & = & \mathtt{N}(B) \end{array}$$

Remark 2.9. *The block-complexity is invariant under congruence, defined by equation $\equiv_b$, because the blocks that can be exchanged this way are exactly the substructures that are not counted in the size of a given structure.*

This complexity measure is simply the sum, for each block δ, of the size of the structure in the scope of δ. We can use this to show that the process of building a derivation by applications of inference rules of JSLb terminates — we call this *proof search* although we are building derivations and not proofs.

Lemma 2.10. *Proof search in* JSLb *is terminating.*

Proof. Given some functional structure A, if we apply any inference rule of JSLb on A other than the contraction crr rule, we obtain a functional structure B such that $\mathtt{C}(B) < \mathtt{C}(A)$, as can be checked for the rules xr, wr, ex and sr.

In the case of crr, we can observe that one positive occurrence of an atom has been replaced with an occurrence of some fresh atom. We define for any functional structure F a measure $\mathtt{M}(F)$ as the multiset of $|F|^+_d$ for all $d \in F$, under multiset ordering, and with crr we have $\mathtt{M}(B) < \mathtt{M}(A)$ since $|B|^+_e < |A|^+_e$ for some $e \in A$, and the introduced atom has a multiplicity of 1 in B.

Finally, we can use an induction on the pair $(\mathtt{M}(A), \mathtt{C}(A))$, under lexicographic order, and we reach the base case with a structure G such that no block δ appears in G. This implies that no rule of JSLb can be applied on G. □

Moreover, we can prove that the rules of JSLb are invertible in JS, so that proof search preserves provability in JS. This means we can use JSLb on a structure A to produce by proof search a B such that A is provable in JS if and only if B is provable in JS. This is expressed in the following lemmas.

Lemma 2.11. *If there is a proof in* JS *of a functional structure* $\xi\{(B \to a) \to a\}$*, then there is a proof in* JS *for* $\xi\{B\}$*.*

Proof. By induction on the height of some given proof $\mathscr{D}$ in JS of $\xi\{(B \to a) \to a\}$, we build a proof for the structure $\xi\{B\}$. If $\mathscr{D}$ has height 2, it uses two identities on $((b \to b) \to a) \to a$, and we use the identity on $b \to b$ only. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this occurrence of $(B \to a) \to a$, we can rewrite the conclusion into $\xi\{B\}$, and use the induction hypothesis to rewrite the premise accordingly — and if r only affects a structure inside this occurrence of B, we proceed the same way.

2. If r is a contraction c on this occurrence of $B \to a$, then one copy of $B \to a$ is erased by a weakening in the proof, and we can rewrite the proof to remove this copy, by replacing all switch instances moving material in the copy of B by weakenings — and obtain a proof of at most the same height, so that we can use the induction hypothesis.

3. If r is a switch s moving a structure D on the left of $B \to a$, we can remove it and go on by induction hypothesis:

$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to a\}}{\xi\{E \to ((B \to a) \to a)\}} \quad \longrightarrow \quad \xi\{E \to B\}$$

Notice that there can be no weakening on this $B \to a$ since there would be no a left in negative position to complete the proof. □

Lemma 2.12. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to a) \to C\}$*, and* $|C|^+_a = 0$*, then there is a proof in* JS *for the structure* $\xi\{C\}$*.*

Proof. By induction on the height of a given proof $\mathscr{D}$ in JS of $\xi\{(B \to a) \to C\}$, we build a proof for the structure $\xi\{C\}$. If $\mathscr{D}$ has height 2, it uses a weakening and an identity on $(B \to a) \to (c \to c)$, and we use the identity on $c \to c$ only to conclude.

In the general case, we use a case analysis on the bottommost rule instance r in the given proof $\mathscr{D}$:

1. If r does not affect this occurrence of $B \to a$, we rewrite the conclusion into $\xi\{C\}$ and use the induction hypothesis to rewrite the premise accordingly.

2. If r only affects a structure inside this occurrence of B, we can rewrite the conclusion into $\xi\{C\}$, and then we use the induction hypothesis to rewrite the premise, as in the previous case.

3. If r is a weakening w on this occurrence of $B \to a$, we can remove it in the conclusion and the result is immediate.

4. If r is a contraction c on this occurrence of $B \to a$, we rewrite the conclusion into $\xi\{C\}$ and then we use twice the induction hypothesis, which is possible since the first proof obtained has at most the same height as the original.

5. If r is a switch s moving a structure E on the left of $B \to a$, we replace it by a weakening, as shown below:

$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to C\}}{\xi\{E \to ((B \to a) \to C)\}} \quad\longrightarrow\quad \mathsf{w}\,\frac{\xi\{C\}}{\xi\{E \to C\}}$$

   and we can go on by induction hypothesis.

Therefore, in any case we can build the expected proof of $\xi\{C\}$. □

Lemma 2.13. *If there is a proof in the* JS *system of some functional structure of the shape* $\xi\{(B \to a) \to ((C \to L) \to D)\}$*, with* $|C|_a^+ \geq 1$ *and* $|D|_a^+ = 0$*, then there is a proof in* JS *for the structure* $\xi\{((((B \to a) \to C) \to L) \to D)\}$.

Proof. By induction on the height of a proof $\mathscr{D}$ of $\xi\{(B \to a) \to ((C \to L) \to D)\}$ in JS translated to the JS′ system, we build a proof of at most the same height for $\xi\{((((B \to a) \to C) \to L) \to D)\}$. In the base case, $\mathscr{D}$ has height 3 and it uses on $((c \to c) \to a) \to ((a \to b) \to b)$ three identities, so that we can do the same. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this occurrence of $B \to a$, we rewrite the conclusion into $\xi\{((((B \to a) \to C) \to L) \to D)\}$ and use the induction hypothesis to rewrite the premise.

2. If r only affects a structure inside this occurrence of B, we can rewrite the conclusion again, and use the induction hypothesis.

3. If r is a weakening w on this occurrence of $B \to a$, we rewrite the conclusion again and use a weakening.

4. If r is a contraction c on this occurrence of $B \to a$, we rewrite the conclusion again and use twice the induction hypothesis, which is possible since the proof obtained the first time has at most the same height as the original.

5. If r is a switch s moving a structure E on the left of $B \to a$, we replace it by a super-switch, as shown below:

$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to ((C \to L) \to D)\}}{\xi\{E \to ((B \to a) \to ((C \to L) \to D))\}}$$

$$\longrightarrow \quad \mathsf{ss}\,\frac{\xi\{((((E \to B) \to a) \to C) \to L) \to D\}}{\xi\{E \to ((((B \to a) \to C) \to L) \to D)\}}$$

and then we go on by induction hypothesis. The case of a super-switch is treated exactly the same way.

In the end, we have a new proof in JS' that we can turn into a proof of the same structure in JS by expanding super-switches. □

Lemma 2.14. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to a) \to C\}$*, with* $|C|_a^+ \geq 2$*, there is a proof in* JS *for* $\xi\{(B \to d) \to ((B \to a) \to C_{[d/a]})\}$.

Proof. We proceed by induction on the pair $(|C|_a^+, h)$, under lexicographic order, where h is the height of the proof $\mathscr{D}$ in JS of $\xi\{(B \to a) \to C\}$, we build a proof for the structure $\xi\{(B \to d) \to ((B \to a) \to C_{[d/a]})\}$. By hypothesis, there is always at least two occurrences of a in C and we can use a case analysis on the bottommost rule instance r in the given proof $\mathscr{D}$:

1. If r does not affect this occurrence of $B \to a$ and does not erase or duplicate the occurrence of a replaced by d in $C_{[d/a]}$, or affects only B, we rewrite the conclusion into the expected one and use the induction hypothesis to rewrite the premise accordingly, since h has decreased.

2. If r is a weakening w erasing the occurrence of a replaced by d in $C_{[d/a]}$, then we simply rewrite the conclusion and introduce a weakening on $B \to d$ at the bottom of the proof.

3. If r is a contraction c duplicating the occurrence of a replaced by d in $C_{[d/a]}$, then we apply the induction hypothesis twice on the proof above r to rewrite the premise — this is possible because $|C|_a^+$ decreased after the first rewriting — and introduce an additional contraction on $B \to d$.

4. If r is a weakening w on this occurrence of $B \to a$, we rewrite the conclusion and insert an additional weakening to erase the structure $B \to d$, while if it is a contraction on $B \to a$, we immediately use the induction hypothesis.

5. If r is a switch s moving a structure E on the left of $B \to a$, we introduce a contraction and a switch, as follows, and go on by induction hypothesis:

$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to C\}}{\xi\{E \to ((B \to a) \to C)\}} \longrightarrow \begin{array}{c} \mathsf{s}^* \dfrac{\xi\{(((E \to B) \to d) \to (E \to B) \to a) \to C_{[d/a]}\}}{\xi\{E \to E \to ((B \to d) \to (B \to a) \to C_{[d/a]})\}} \\ \mathsf{c}\,\dfrac{}{\xi\{E \to ((B \to d) \to (B \to a) \to C_{[d/a]})\}} \end{array}$$

□

Lemma 2.15. *If there is a proof of a functional structure* $\xi\{(B \to a) \to (C \to D)\}$ *in* JS *with* $|C|_a^+ = 0$ *and* $|D|_a^+ \geq 1$*, there is a proof in* JS *for* $\xi\{C \to ((B \to a) \to D)\}$.

Proof. This corresponds to the use of the congruence in the JS system, namely the equation $\equiv_a$, so that this is immediate. □

We can use all these lemmas to prove that the proof search process in JSLb has the interesting property of preserving provability, which will be a crucial argument in our partial completeness result.

Lemma 2.16. *For any structures A and B, if there is a derivation from A to B in the* JSLb *system and B is provable in* JS*, then A is provable in* JS.

Proof. By induction on the height of the given derivation $\mathscr{D}$ from A to B in JSLb. If $\mathscr{D}$ has height 0, the result is immediate since A is B. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$, to prove that if there is a proof of its conclusion A in the JS system, there is also a proof of its premise, by either Lemma 2.11, Lemma 2.12, Lemma 2.14, Lemma 2.15 or Lemma 2.13. □

2.3 Closing JSLb Proofs

The rules of JSLb form a subsystem of JSL that cannot be used in general for the construction of complete proofs, because of the strong restrictions imposed on the use of weakenings. In order to establish a partial completeness result, we now show how to close a proof by dealing with weakenings that cannot be performed in JSLb or in JSL, to recover the provability of the JS system.

Weakening matchers. We consider the following rule, which corresponds to a case of weakening forbidden in JSL:

$$\mathsf{uw}\ \frac{A}{(B \to \top) \to A}$$

Then, we can show that this inference rule is invertible, in the sense that using it during the proof search process cannot turn some provable structure into another structure that is not provable — reading from conclusion to premise.

Lemma 2.17. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to \top) \to A\}$*, then there is a proof of* $\xi\{A\}$ *in* JS.

Proof. By induction on the given proof $\mathscr{D}$ in JS, we build a proof of $\xi\{A\}$, preserving the upper bound on the height of the proof. If $\mathscr{D}$ has height 1, it uses a weakening on $(B \to \top) \to \top$, and the result is immediate. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this $B \to \top$ occurrence, we remove it from the conclusion and use the induction hypothesis to rewrite the premise accordingly.

2. If r only affects a structure inside this occurrence of B, we can remove this instance and use the induction hypothesis.

3. If r is a weakening w on this occurrence of $B \to \top$, the result is immediate.

4. If r is a contraction c on this occurrence of $B \to \top$, we rewrite the conclusion and we use twice the induction hypothesis, which is possible since the proof obtained the first time has at most the same height as the original.

5. If r is a switch s moving a structure C on the left of B, we can replace it by a weakening, as shown below, and go on by induction hypothesis:

$$\mathsf{s}\,\frac{\xi\{((C \to B) \to \top) \to A\}}{\xi\{C \to ((B \to \top) \to A)\}} \quad\longrightarrow\quad \mathsf{w}\,\frac{\xi\{A\}}{\xi\{C \to A\}}$$

In this analysis, there is no need to consider the case where r is an instance of the identity x used on $(B \to \top) \to \top$, since we can always remove it from any proof in JS by using Proposition 2.5. □

Closing the proof. We consider a functional structure where no block and no matcher appears, and observe that it is of the shape $b_1 \to \cdots \to b_n \to a$. Such a structure is provable if and only if there is an i such that b_i is a, and we can use the following inference rule, not to be applied inside a context:

$$\mathsf{xw}\,\frac{\top}{b_1 \to \cdots \to a \to \cdots \to b_n \to a}$$

which can be applied on a structure of this shape if and only if it is provable, since it is equivalent to many weakenings and one identity instance. Now, we can glue the pieces together to produce a proof of weak completeness, which states that given a functional structure A provable in JS, although there is no proof of A in JSLb nor in JSLd, there is a derivation from a structure B to A in JSLb, such that B can easily be mechanically checked.

Theorem 2.18 (Weak completeness of JSLb)**.** *For any given functional structure A, if there is a proof of A in JS, then there is a structure B such that there is a derivation from B to A in JSLb and a proof of B in $\{\mathsf{uw}, \mathsf{xw}\}$.*

Proof. As a first step, we apply Lemma 2.10 to produce by proof search a derivation $\mathscr{D}_1$ from some structure A_1 to A in the JLSb system. Notice that there can be no block — that is, structures of the shape $(B \to c)$ in negative position — in the structure A_1 since that would imply that at least one inference rule of JSLb could be applied. Moreover, by Lemma 2.16 we know that if A is provable in JS, then A_1 is provable in JS too. Then, we apply as much as possible the uw rule on A_1 to produce a derivation $\mathscr{D}_2$ from a structure A_2 with no matchers — that is, structures of the shape $(B \to \top)$ in negative position — to A_1. By Lemma 2.17 we know that if A_1 is provable in JS, then A_2 is provable in JS too. Finally, if A_2 is provable in JS and does not contain matchers, then we can use one instance of xw to build a proof of the shape described in (11), where the expected derivation from B to A in JSLb is $\mathscr{D}_1$, since this A_1 is provable in $\{\mathsf{uw}, \mathsf{xw}\}$. □

Corollary 2.19 (Weak completeness of JSLd)**.** *For any given functional structure A, if there is a proof of A in* JS*, then there is a structure B such that there is a derivation from B to A in* JSLd *and a proof of B in* {uw, xw}.

This result tells us that JSLb is indeed a sensible system to deal with functional structures, and therefore JSLd can also be used. The problem of proving functional structures in the JSLd system is more subtle that in JSLb, since it allows the use of the xsu rule which is not invertible, while we can avoid using the uw rule to get a complete proof in some cases. This is not always possible, as shown by the following example proof:

$$\mathsf{uw}\,\dfrac{\mathsf{xw}\,\dfrac{\top}{a \to a}}{(B \to \top) \to (a \to a)}$$

In this case, we cannot use the xsu rule if the structure B is not provable in JS, while the complete structure is provable in JS. The JSLd system allows to use the minimal amount of instances of uw to get a proof in JS, but there is no way in general to build a proof from JSLd and the xw rule only. The class of structures that can be proved using JSLd and xw only is more behaved than general functional structures, since the structure B in a structure of the shape $(B \to \top) \to C$ is not logically *relevant*.

3 Encoding Reduction in Proof Search

We will now consider a λ-calculus with explicit substitutions called $\lambda\mathtt{s}$ [KR11], and show how its terms can be encoded into logical structures, so that the process of building a derivation in JLSd will simulate the process of applying reduction rules in the $\lambda\mathtt{s}$-calculus. We say *proof search*, although we are not building complete proofs but rather open derivations, to emphasize the relation of this work with the usual *proof-search-as-computation* paradigm [MNPS91].

It is a direct approach, based on incomplete derivations of the proof system, in the sense that if two given structures A and B represent programs such that A can be reduced to B through the operational semantics of the language, there is a derivation from B to A rather than a proof of $A \to B$, as we will see.

Our λ-calculus is very similar to many other calculi with explicit substitutions in the literature [Ren11], and in particular it borrows its handling of duplication using a linear renaming operation from the *structural λ-calculus* [AK10]. The syntax of $\lambda\mathtt{s}$-terms can be defined by the following grammar:

$$t, u \;::=\; x \mid \lambda x.t \mid t\ u \mid t[x \leftarrow u]$$

where the object $[x \leftarrow u]$, which is called an *explicit substitution*, is a binder for the variable x, so that x is bound in $t[x \leftarrow u]$. Moreover, terms are considered *modulo* α-conversion, so that a variable is always bound at most once in any $\lambda\mathtt{s}$-term. We will need to count the use of variables in a term, using the following definition.

$$
\begin{array}{rcll}
(\lambda x.t)\,u & \longrightarrow_{\mathtt{B}} & t[x \leftarrow u] & \\
x[x \leftarrow u] & \longrightarrow_{\mathtt{var}} & u & \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{not}} & t & \text{if } |t|_x = 0 \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{dup}} & t_{[y/x]}[x \leftarrow u][y \leftarrow u] & \text{if } |t|_x \geq 2 \\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] & \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apl}} & t[x \leftarrow u]\ v & \text{if } |v|_x = 0 \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apr}} & t\ v[x \leftarrow u] & \text{if } |t|_x = 0 \\
t[y \leftarrow v][x \leftarrow u] & \longrightarrow_{\mathtt{cmp}} & t[y \leftarrow v[x \leftarrow u]] & \text{if } |t|_x = 0, |v|_x \geq 1
\end{array}
$$

$$
t[y \leftarrow v][x \leftarrow u] \equiv t[x \leftarrow u][y \leftarrow v] \qquad \text{if } |v|_x = |u|_y = 0
$$

Figure 4: Reduction rules and equation for the $\lambda\mathtt{s}$-calculus

Definition 3.1. *The* multiplicity *of a variable x in a term t, denoted by $|t|_x$, is the number of occurrences of the variable x in the term t, not including uses as binder.*

The reduction rules defining the operational behaviour of the $\lambda\mathtt{s}$-calculus are shown in Figure 4, where the construction $t_{[y/x]}$ denotes the term t where exactly one occurrence of x has been replaced with y. Notice that in the `dup` rule, the new variable y must of course be fresh to avoid capture by some binder. This system of reduction rules is similar to many other calculi and was presented in Chapter 2.

We can now define an encoding of $\lambda\mathtt{s}$-terms into structures. For that, we need to consider a bijection between logical atoms and variables in the calculus, so that to any variable x corresponds an atom also denoted by x.

Definition 3.2 (Encoding of $\lambda\mathtt{s}$)**.** *The encoding $[\![\cdot]\!]_\lambda$ from $\lambda\mathtt{s}$-terms into structures of the* JSLd *system is defined as follows:*

$$
\begin{array}{rcl}
[\![x]\!]_\lambda & = & x \\
[\![\lambda x.t]\!]_\lambda & = & x \to [\![t]\!]_\lambda \\
[\![t\ u]\!]_\lambda & = & ([\![u]\!]_\lambda \to \top) \to [\![t]\!]_\lambda \\
[\![t[x \leftarrow u]]\!]_\lambda & = & ([\![u]\!]_\lambda \to x) \to [\![t]\!]_\lambda
\end{array}
$$

Notice that through this encoding, $\lambda\mathtt{s}$-terms correspond to functional structures only, as they were defined to be used with the JSLd system — this is of course the reason why we restricted the rules of JS this way. Moreover, the equation on terms in $\lambda\mathtt{s}$ allowing to exchange unrelated explicit substitutions exactly corresponds to the equation $\equiv_b$ used on functional structures.

Remark 3.3. *It is easy to observe that the encoding $[\![\cdot]\!]_\lambda$ defines a bijection between $\lambda\mathtt{s}$-terms and functional structures. Indeed, each shape of structure defined in the grammar for restricted structures corresponds to exactly one construction in the terms syntax, and the extra restrictions for functional structures correspond to the writing of a $\lambda\mathtt{s}$-term with a correct scope structure α-converted to avoid repetition of bindings on the same name.*

3.1 Computational Adequacy

We now state the theorem establishing the correspondence, at the computational level, between the operational behaviour of the $\lambda\mathtt{s}$-calculus and the behaviour of proof search in the JSLd system, where the $\longrightarrow_{\lambda\mathtt{s}}$ relation is the reduction defined by the rules of Figure 4, $\longrightarrow^*_{\lambda\mathtt{s}}$ is its reflexive and transitive closure, and $\longrightarrow_{\mathtt{s}}$ is the same, where the B rule is not used.

Theorem 3.4 (Computational adequacy of JSLd)**.** *For any given $\lambda\mathtt{s}$-terms t and u, there is a derivation from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in the* JSLd *system if and only if $t \longrightarrow^*_{\lambda\mathtt{s}} u$.*

Proof. By induction on the reduction steps from t to u. If the reduction path is empty, t and u are the same and the result is trivial because the encoding $[\![\cdot]\!]_\lambda$ is uniquely defined. In the general case, we consider the first step in the reduction, and use the induction hypothesis on the rest of the reduction. We thus simply have to check that there is an inference rule instance in JSLd with premise $[\![u]\!]_\lambda$ and conclusion $[\![t]\!]_\lambda$ if and only if we have $t \longrightarrow_{\lambda\mathtt{s}} u$. This is immediately done by observing that each reduction rule corresponds exactly to one case of application of an inference rule of JSLd:

$$\mathsf{xsu}\,\frac{(B \to a) \to C}{(B \to \top) \to (a \to C)} \quad\longleftrightarrow\quad \mathtt{B}\,\frac{t[x \leftarrow u]}{(\lambda x.t)\,u}$$

$$\mathsf{xr}\,\frac{B}{(B \to a) \to a} \quad\longleftrightarrow\quad \mathtt{var}\,\frac{u}{x[x \leftarrow u]}$$

$$\mathsf{wr}\,\frac{C}{(B \to a) \to C} \quad\longleftrightarrow\quad \mathtt{not}\,\frac{t}{t[x \leftarrow u]}$$

$$\mathsf{cr}\,\frac{(B \to d) \to ((B \to a) \to C_{[d/a]})}{(B \to a) \to C} \quad\longleftrightarrow\quad \mathtt{dup}\,\frac{t_{[y/x]}[x \leftarrow u][y \leftarrow u]}{t[x \leftarrow u]}$$

$$\mathsf{ex}\,\frac{c \to ((B \to a) \to D)}{(B \to a) \to (c \to D)} \quad\longleftrightarrow\quad \mathtt{lam}\,\frac{\lambda y.t[x \leftarrow u]}{(\lambda y.t)[x \leftarrow u]}$$

$$\mathsf{ex}\,\frac{(C \to \top) \to ((B \to a) \to D)}{(B \to a) \to ((C \to \top) \to D)} \quad\longleftrightarrow\quad \mathtt{apl}\,\frac{t[x \leftarrow u]\ v}{(t\ v)[x \leftarrow u]}$$

$$\mathsf{sr}\,\frac{(((B \to a) \to C) \to \top) \to D}{(B \to a) \to ((C \to \top) \to D)} \quad\longleftrightarrow\quad \mathtt{apr}\,\frac{t\ v[x \leftarrow u]}{(t\ v)[x \leftarrow u]}$$

$$\mathsf{sr}\,\frac{(((B \to a) \to C) \to e) \to D}{(B \to a) \to ((C \to e) \to D)} \quad\longleftrightarrow\quad \mathtt{cmp}\,\frac{t[y \leftarrow v[x \leftarrow u]]}{t[y \leftarrow v][x \leftarrow u]}$$

Notice that the conditions on the multiplicity of variables in the reduction rules for $\lambda\mathtt{s}$-terms exactly match the restrictions that were imposed on inference rules to define JSLd from JSL. □

This result establishes a tight connection between our restricted intuitionistic system and the λs-calculus. The interesting point is then that a theorem that we can prove on derivations of the JSLd system on the logical side also holds in its *computational* form on reduction paths in the λs-calculus. As an example, we can prove that the subsystem $\longrightarrow_{\mathtt{s}}$ terminates.

Proposition 3.5. *The reduction subsystem* $\longrightarrow_{\mathtt{s}}$ *terminates.*

Proof. This is a direct corollary of Lemma 2.10, considering derivations of JSLb as reduction paths in the $\longrightarrow_{\mathtt{s}}$ subsystem through the correspondence defined by Theorem 3.4. □

The other way around, if we have some result on reduction in the λs-calculus, we can directly transpose this result to the logical side. For example, confluence can be proved for the the $\longrightarrow_{\lambda\mathtt{s}}$ rewrite system, and we could mimick the proof to obtain a similar result on derivations of the JSLd system. We would thus obtain a proof of the following proposition.

Proposition 3.6. *For any structures A, B and C, if there are derivations from B to A and from C to A in* JSLd*, then there is a structure D such that there are derivations from D to B and from D to C in* JSLd*.*

Moreover, we observed that the class of structures that can be proven by proof search in JSLd without using uw is more interesting than functional structures, since this rule does not correspond to a valid rewriting on λs-terms. Also notice that the conclusion of the xw rule, where no structure of the shape $B \to \top$ appears, is exactly a λs-term in normal form. From this we can derive a characterisation of *weakly normalising* terms.

Theorem 3.7. *A λs-term t is weakly normalising if and only if there is a proof of* $[\![t]\!]_\lambda$ *in the* JSLd ∪ {xw} *system.*

Proof. First, if the term t is weakly normalising, then there is a u in normal form with $t \longrightarrow^*_{\lambda\mathtt{s}} u$, and by Theorem 3.4 we have a derivation $\mathscr{D}$ from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in JSLd. We can thus use the xw rule on $[\![u]\!]_\lambda$ to produce a proof of $[\![t]\!]_\lambda$. Then, if there is a proof of $[\![t]\!]_\lambda$ in JSLd ∪ {xw}, we have such a derivation $\mathscr{D}$ and we can use Theorem 3.4 the other way around to get a term u in normal form such that we have $t \longrightarrow^*_{\lambda\mathtt{s}} u$. □

It would therefore be interesting to learn more about this class of structures, inside the logic, to get insights on weakly normalising terms in the λs-calculus. Furthermore, an important class is the one of *strongly normalising* terms, for which any reduction path reaches a normal form, and which are often characterised using a type system. It is not clear whether this class could be characterised through a particular variant of the JSLd system. An interesting problem would therefore be to define an efficient procedure to decide whether a given structure is provable in this system, to check if some λs-term is strongly normalising without computing its typing derivation.

In particular, this means that we could ensure termination of a program without knowing its type: if this can be done efficiently using an algorithm such as a variant of resolution for a fragment of intuitionistic logic, this is interesting, but it does not ensure that the composition of two well-behaved programs is well-behaved. It is thus unclear what kind of mechanism could take the role of typing in this setting.

The tight correspondence established by Theorem 3.4 allows direct reasoning on reduction paths in the $\lambda\mathtt{s}$-calculus, where each step can be handled separately. Therefore, if we have a trace of computation corresponding to the reduction of a term t to some term u and another trace for the reduction from u to some term v, composing these traces is immediate and provides a trace of computation from t to v. Contexts can also be handled in a very natural way, as it is done in the calculus of structures.

3.2 Search Strategies and Evaluation Order

In the setting established through computational adequacy, permutations of rule instances, and transformations on derivations in the JSLd proof system allow for a reorganisation of a reduction path between two given $\lambda\mathtt{s}$-terms. Indeed, the order in which instances appear in a derivation corresponds to an order in the treatment of the redexes of a $\lambda\mathtt{s}$-term, so that the choice of a proof search strategy in JSLd is equivalent to the choice of an evaluation order for a given term. There are several interesting observations regarding this correspondence:

- The relative order of xsu instances in a derivation corresponds to the order in which β-redexes are turned into explicit substitutions, and a xsu instance cannot always be permuted down, since new β-redexes can be created during reduction.
- A trivial permutation corresponds to the situation where two redexes can be treated independently, which means that there is local confluence — because we can apply the two reduction rules in any order.
- The confluence result in the $\lambda\mathtt{s}$-calculus implies that there is always a form of permutation possible between rule instances of a derivation in JSLd, which might require to introduce a new sequence of reductions, corresponding to a subderivation introduced to make the premise of one instance match the conclusion of the other.

Example 3.8. *Below are shown two sequences of rewriting, presented as derivations of* JSLd *but where structures are written as λ-terms. These two derivations correspond to the permutation of an* ex *instance below a* xsu *instance, and to the two reduction sequences to obtain the term* $t[y \leftarrow v][x \leftarrow u]$ *from the term* $((\lambda x.t)\,u)[y \leftarrow v]$.

$$\mathsf{ex}\,\dfrac{\mathsf{xsu}\,\dfrac{t[y \leftarrow v][x \leftarrow u]}{t[x \leftarrow u][y \leftarrow v]}}{((\lambda x.t)\,u)[y \leftarrow v]} \qquad \longleftrightarrow \qquad \mathsf{xsu}\,\dfrac{\mathsf{ex}\,\dfrac{t[y \leftarrow v][x \leftarrow u]}{(\lambda x.t)[y \leftarrow v]\,u}}{((\lambda x.t)\,u)[y \leftarrow v]}$$

$$\mathsf{xr}\ \frac{B}{(B \to a) \to \Downarrow a} \qquad \mathsf{xsu}\ \frac{(B \to a) \to \Downarrow C}{(B \to \top) \to (a \to C)}$$

$$\mathsf{wr}\ \frac{A}{\delta \to \Downarrow A} \qquad \mathsf{cr}\ \frac{\delta \to \Downarrow (\delta \to \Downarrow A)}{\delta \to \Downarrow A}$$

$$\mathsf{ex}\ \frac{A \to (\delta \to \Downarrow B)}{\delta \to \Downarrow (A \to B)} \qquad \mathsf{sr}\ \frac{((\delta \to \Downarrow A) \to L) \to B}{\delta \to \Downarrow ((A \to L) \to B)}$$

Figure 5: Inference rules for system JLSn

4 Focused Proof Search and Strategies

Following our methodology, there is a direct connection between the cases where an inference rule can be applied on a structure and the reduction strategy that is implemented by the system for $\lambda\mathtt{s}$. Indeed, we can use a variant of JSLd not using extra conditions on the multiplicity of atoms in the rules, but this would induce highly non-deterministic reduction rules for the $\lambda\mathtt{s}$-calculus — and it would change the properties of the calculus. We made the reduction system deterministic by imposing a strategy in the sense that one reduction rule can be applied in only one way, but there is still a non-deterministic dimension in the choice of which redex to pick and reduce, given a $\lambda\mathtt{s}$-terms with several redexes. We now address the question of this choice, which means choosing a reduction strategy.

4.1 The JSLn Focused System

The JSLd system can be restricted further by using annotations on structures in the spirit of focusing [And92], and this can be interpreted on the computational side as restricting the choice of the next redex to be rewritten. This restriction is similar to a standard formulation of *focusing* for intuitionistic logic [LM07], although the deep inference setting used here does not allow the exact same treatment. It should be noted that focusing in the calculus of structures cannot be immediately defined through a translation of usual focusing in the sequent calculus, as described in Chapter 7. For example, there are no separations between different branches, and here we will duplicate focusing annotations, moving copies to different parts of the structure, corresponding to branches. The approach used on JSL is not the same as the one developped in Chapter 7 — and we could discuss the status of focused system of the variant of JSL presented here, but the idea is simply to reduce the search space using annotations.

The inference rules for the JSLn system are shown in Figure 5, where the syntax of structures is extended with annotations, so that $\xi\{\Downarrow A\}$ is a valid structure for any $\xi\{\ \}$ and A — annotations are only allowed in positive position. Then, the *decision*

action is embedded inside the xsu rule, which picks a structure on the right of a block of the shape $B \to a$ and forces to move this block inside this structure until its contents, in B, are released by the xr rule. This sequence of rule applications guided by annotations is called a focused *phase*, and we are mainly interested in the structures at the borders of such a phase.

Definition 4.1. *A functional structure B is said to be* basic *if all the structures in negative position inside B are either atoms or matchers — it contains no block, as $C \to a$ in negative position.*

The point of the JSLn system is then to handle basic structures by choosing a redex for the xsu rule, to introduce a block $B \to a$ through this rule, and then maximally use the JSLb subsystem to produce a new basic structure, by removing all the remaining blocks. In particular, we add the restriction that the rule xsu is not used on a structure that already contains a focus annotation. It is easy to see that any basic formula corresponds through the encoding $[\![\cdot]\!]_\lambda$ to some pure λ-term, which is a λs-term without explicit substitutions. We can therefore consider the standard rule for β-reduction in the pure λ-calculus:

$$(\lambda x.t)\, u \longrightarrow_\beta t\{u/x\}$$

and show that there is a computational adequacy result between JSLn and this simple rewriting system, through the same $[\![\cdot]\!]_\lambda$ encoding as before. As first step, we need a lemma explaining the effect of a focusing phase on a structure. In the following, we denote by $\xi\{A\}^+$ a context with several holes filled with the structure A, and by $\xi\{A\}^*$ a context with zero or more holes filled with A. Moreover, the notation $\xi\{a\}^*$ implicitly means there is no occurrence of the atom a in $\xi\{\ \}^*$ except in its holes.

Lemma 4.2. *Given a functional structure where no annotation appears, of the shape $\xi\{(B \to \top) \to (a \to \zeta\{a\}^*)\}$, there is a derivation from $\xi\{\zeta\{B\}^*\}$ to this structure in the* JSLn *system.*

Proof. Given a functional structure of the shape $\xi\{(B \to \top) \to (a \to \zeta\{a\}^*)\}$, if there are no annotations inside it we prove that there is a derivation $\mathscr{D}$ in JSLn such that we have the following situation:

$$\mathsf{xsu}\ \dfrac{\begin{array}{c}\xi\{\zeta\{B\}^*\}\\ \mathscr{D}\,\|\\ \xi\{(B \to a) \to \Downarrow \zeta\{a\}^*\}\end{array}}{\xi\{(B \to \top) \to (a \to \zeta\{a\}^*)\}}$$

We proceed by induction on the size of the context $\zeta\{\ \}^*$ to prove that there is such a $\mathscr{D}$, using a case analysis on its toplevel shape:

1. If $\zeta\{\ \}^*$ is $\{\ \}$, then we can use an identity rule, as shown below, and we are done since we have our derivation $\mathscr{D}$.

$$\mathsf{xr}\ \dfrac{\xi\{B\}}{\xi\{(B \to a) \to \Downarrow a\}}$$

2. If $\zeta\{\ \}^*$ is C, where the atom a does not appear in C, then we can use a weakening rule, as shown below, and we are also done with the induction:

$$\mathsf{wr}\ \frac{\xi\{C\}}{\xi\{(B \to a) \to \Downarrow C\}}$$

3. If $\zeta\{\ \}^*$ is $(C \to L) \to \theta\{\ \}^+$, then we can use an exchange rule, since the atom a does not appear in C, and then go on by induction hypothesis:

$$\mathsf{ex}\ \frac{\xi\{(C \to L) \to ((B \to a) \to \Downarrow \theta\{a\}^+)\}}{\xi\{(B \to a) \to \Downarrow ((C \to L) \to \theta\{a\}^+)\}}$$

4. If $\zeta\{\ \}^*$ is $(\theta\{\ \}^+ \to L) \to C$, then we can use a switch rule, since the atom a does not appear in C, and then go on by induction hypothesis:

$$\mathsf{sr}\ \frac{\xi\{(((B \to a) \to \Downarrow \theta\{a\}^+) \to L) \to C\}}{\xi\{(B \to a) \to \Downarrow ((\theta\{a\}^+ \to L) \to C)\}}$$

5. If $\zeta\{\ \}^*$ is $(\theta\{\ \}^+ \to L) \to \theta'\{a\}^+$, then we have to use both the exchange and switch rules, after using a contraction rule to duplicate the block $B \to a$, and then go on by using twice the induction hypothesis:

$$\mathsf{cr}\ \frac{\mathsf{sr}\ \dfrac{\mathsf{ex}\ \dfrac{\xi\{(((B \to a) \to \Downarrow \theta\{a\}^+) \to L) \to ((B \to a) \to \Downarrow \theta'\{a\}^+)\}}{\xi\{(B \to a) \to \Downarrow (((B \to a) \to \Downarrow \theta\{a\}^+) \to L) \to \theta'\{a\}^+\}}}{\xi\{(B \to a) \to \Downarrow ((B \to a) \to \Downarrow ((\theta\{a\}^+ \to L) \to \theta'\{a\}^+))\}}}{\xi\{(B \to a) \to \Downarrow ((\theta\{a\}^+ \to L) \to \theta'\{a\}^+)\}}$$

Notice that if the context $\zeta\{\ \}^*$ had no hole, the weakening rule is applied and no replacement is performed. □

The JSLn system is a simple restriction of the JSLd system presented in the previous section, and we could show that it is complete with respect to JSLd in the sense that if there is a derivation from A to B in JSLd then there is also a derivation with same premise and conclusion in JSLn, provided that A and B are basic. This will be proved in an external way, using the computational interpretation of proof search in these systems.

4.2 Focusing as Big-step Computation

The effect of the modifications of JSL leading to the focused variant JSLn on proof search is the reduction of choices among inference rules that can be applied on a given structure. In particular a sequence of rule instances pushing a block $B \to a$ towards other occurrences of a is » *grouped* « in this system, in the sense that this sequence cannot be interleaved with another such sequence. From a computational viewpoint, this corresponds to the separation of the treatment of different explicit substitutions, and this is the way to mimick the plain β-reduction strategy of the pure λ-calculus.

Theorem 4.3 (Computational adequacy of JSLn)**.** *For any two λ-terms t and u, there is a derivation from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in the* JSLn *system if and only if $t \longrightarrow_\beta^* u$.*

Proof. By induction on the reduction steps from t to u. If the reduction path is empty, t and u are the same and the result is trivial because the encoding $[\![\cdot]\!]_\lambda$ is uniquely defined. In the general case, we consider the first step in the reduction, and use the induction hypothesis on the rest of the reduction. We thus simply have to check that there is an inference rule instance in JSLn with premise $[\![u]\!]_\lambda$ and conclusion $[\![t]\!]_\lambda$ if and only if we have $t \longrightarrow_\beta u$. To do it, we consider a particular redex, such that t is the term $C[(\lambda x.v)\ w]$ for some context $C[-]$, and show that u is $C[v\{w/x\}]$ if and only if there is a derivation from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in JSLn. More precisely, this derivation exactly consists of one phase, where the bottommost rule instance is an instance of xsu with premise $([\![w]\!]_\lambda \to x) \to \Downarrow [\![v]\!]_\lambda$, and the premise of the phase is some basic structure which must be $[\![u]\!]_\lambda$, as a direct consequence of Lemma 4.2. □

This also provides information on the λs-calculus, since the steps used in JSLn to push a block correspond to the rules pushing explicit substitutions.

Corollary 4.4 (Full composition)**.** *For any t and u, we have $t[x \leftarrow u] \longrightarrow_{\mathtt{s}}^* t\{u/x\}$.*

Proof. This is a corollary of Lemma 4.2 when considered through the encoding $[\![\cdot]\!]_\lambda$ using Theorem 4.3. Indeed, the term $t\{u/x\}$ corresponds through $[\![\cdot]\!]_\lambda$ to the structure $[\![t]\!]_\lambda$ where all occurrences of the atom x are replaced with the structure $[\![u]\!]_\lambda$. □

This theorem establishes a precise correspondence between one β-reduction big step, performing an implicit substitution inside the term, and a focusing phase in JSLn. We can now express the relation between the big-step reduction in the λ-calculus and the small-step operational behaviour which is implemented in the λs-calculus, and this corresponds to the study of soundness and completeness of JSLn with respect to JSLd.

Theorem 4.5 (Soundness of JSLn)**.** *For two basic structures A and B, if there is a derivation from A to B in* JSLn *then there is a derivation from A to B in* JSLd*.*

Proof. Each inference rule in the JSLn system is actually an inference rule of JSLd with focus annotations. Therefore, we just need to remove annotations in the given derivation from A to B in JSLn to produce a derivation from A to B in JSLd. □

The meaning of this theorem, on the computational side, is that any reduction path from t to u in the restricted reduction system of β-reduction can be replaced with a reduction sequence from t to u in the more general $\longrightarrow_{\lambda\mathtt{s}}$ rewrite system, which is exactly stepwise simulation of β-reduction by explicit substitutions.

Corollary 4.6 (Simulation of β)**.** *For any λ-terms t and u, if $t \longrightarrow_\beta^* u$ then $t \longrightarrow_S^* u$.*

The other direction of the correspondence between JSLd and JSLn is more interesting, since it indicates that the simulation of β-reduction in the $\lambda\mathtt{s}$-calculus does not rely on the use of a reduction system that would not be sensible.

Lemma 4.7. *For any basic structure A, if there is a derivation from A to a structure $\xi\{(B \to c) \to \zeta\{c\}^*\}$ in* JSLd*, there is a derivation of at most the same height from A to $\xi\{\zeta\{B\}^*\}$ in* JSLd*.*

Proof. By induction on the context $\zeta\{\ \}^*$. If $\zeta\{\ \}^*$ is $\{\ \}$, then we can use the xr rule and an induction on the given derivation $\mathscr{D}$ to remove the structure $(B \to c)$ and replace c with B. This is similar to the result of Lemma 2.11, stating the invertibility of the rule xr. In the general case, we can use a case analysis on the shape of $\zeta\{\ \}^*$ and in each case use an induction on the given derivation, as for xr. This induction is a variant of invertibility for the rules of JLSb, which preserves the upper bound on the height of the derivation, and relies on the fact that the given derivation uses these rules because its premise A is a basic structure — and $B \to c$ thus does not appear in A. □

Theorem 4.8 (Completeness of JSLn)**.** *Given any two basic structures A and B, if there is a derivation from A to B in the* JSLd *system, there is a derivation from A to B in the* JSLn *system.*

Proof. By induction on a given derivation $\mathscr{D}$ from A to B in the JSLd system. If $\mathscr{D}$ is of height 0, then A is B and there is a trivial derivation from A to B in JSLn. In the general case, since B is a basic structure, there is no structure of the shape $C \to d$ in negative position in B and the bottommost rule instance must be an instance of the xsu rule, and has a structure of the shape $\xi\{(C \to d) \to \zeta\{d\}^*\}$ as premise. Thus, by Lemma 4.7 there is a derivation $\mathscr{D}_1$ of smaller height than $\mathscr{D}$ from A to $\xi\{\zeta\{C\}^*\}$ in JSLd, and by Lemma 4.2 there is a derivation $\mathscr{D}_2$ from $\xi\{\zeta\{C\}^*\}$ to B in JSLn. We can then apply the induction hypothesis to $\mathscr{D}_1$ to produce a derivation $\mathscr{D}_3$ and the result is the composition of the two derivations $\mathscr{D}_2$ and $\mathscr{D}_3$. □

On the computational side, this theorem says that the $\lambda\mathtt{s}$-calculus is sensible with respect to the standard λ-calculus. Indeed, a way of defining a reduction system that can simulate β-reduction is to add *too many* possible reductions, thus losing all good properties. This is not the case here, since we have stated in this theorem the projection of reduction with explicit substitutions inside β-reduction. Notice that in the following corollary, it is important that the given terms t and u are plain λ-terms, and not terms with explicit substitutions.

Corollary 4.9 (Projection in β)**.** *For λ-terms t and u, if $t \longrightarrow^*_{\lambda\mathtt{s}} u$ then $t \longrightarrow^*_{\beta} u$.*

We have now all the elements to compare the rewrite systems of the standard λ-calculus and the $\lambda\mathtt{s}$-calculus with explicit substitutions in terms of comparisons between the logical systems JSLn and JSLd that we have defined. Moreover, there are many possible restrictions of the JSLd system that correspond to other λ-calculi or various reduction strategies. For example, we could add restrictions on inference rules of JSLn to enforce a normal order evaluation, so that this variant would

correspond exactly to the *call-by-name* weak reduction. Enforcing a *call-by-value* reduction strategy is more complicated, since the required restrictions on inference rules would be less natural, because of the problem of detecting structures that represent values. Notice that using a shallow proof search strategy betrays the idea of using the deep inference methodology, the same way as using a weak reduction strategy *» betrays the very spirit of the λ-calculus «* [Asp98].

Conclusion

Although the deep inference methodology was rather successful in the field of pure structural proof theory since its inception, its extension into the logical foundations of computation has remained underdevelopped. This might be the consequence of the radical departure from the traditional structure of proof objects based on trees, leading to the idea that a computational interpretation of a nested system must be based on an exotic model, far from the well-understood λ-calculus. As this was of course a consequence of the lack of simple, syntactic normalisation procedures for the systems developped for classical, linear and other logics. On the side of proof search, the huge amount of non-determinism introduced by the ability to apply a rule inside a formula is an immediate obstacle to the use of nested formalisms.

The main conclusion of the work presented here is that the dynamics of proof objects in nested systems in the calculus of structures, or in nested sequents, and their computational interpretations, are not necessarily of a different nature than what can be found in the standard, shallow setting of natural deduction and the sequent calculus. First of all, in the basic framework of intuitionistic logic, systems can be designed in nested formalisms that enjoy cut elimination or normalisation, supported by a proof defining a syntactic transformation, based on rewriting. Then, this rewriting procedure can be understood as computation, following the same scheme as in the standard proofs-as-programs approach, through the definition of type systems. These two steps, described in Chapter 3 and Chapter 4 for the logical part, and in Chapter 5 for the computational part, allow to conclude that nested systems can be assigned a standard computational interpretation.

But while we can relate proof systems in the calculus of structures and in nested sequents to conventional computational devices, the particular features of this kind of systems allow to go beyond what can be done with shallow formalisms, without having to start anew with exotic models. Indeed, the refinements observed at the logical level — in the calculus of structures, for example — induce the refinement of the description of computation obtained through the typing methodology. Thus, there is a purpose in relating the nested setting to well-known interpretations, as it allows to improve the expressivity of languages that can be extracted from logical systems. In Chapter 6, we have seen how the calculus of structures can yield a rich treatment of duplication and linearity in the λ-calculus. More importantly, the crucial role of the switch rule has been exposed, through its use as a typing rule for new resource operators introducing a notion of communication in the λ-calculus in a way that cannot fit the shallow typing setting. This supports the idea that nested systems allow to refine computational interpretations of proofs.

There are still many aspects of the interpretation of the switch rule, in various systems, to investigate, and in particular its relation to the embedding of evaluation strategies into λ-terms, and the nature of the communication happening between subterms during reduction. Interestingly enough, this could allow to establish some connection between the usual framework of functional programming and another possible interpretation for deep inference proof systems, where programs would be sequences operations, just as proofs are sequences of rule instances. Indeed, the introduction of the switch, and of the corresponding communication operators, has a direct impact on the possible reduction strategies for the term. A question would be to know how much freedom is left in the choice of the strategy when using the switch, in comparison with the situation of a completely sequential program.

Moreover, systems in the nested formalisms that follow the shape of the sequent calculus offer new possibilities in terms of reduction, although they correspond to more complex variants of the λ-calculus. The novelty in such a system is that the dual of all the basic inference rules, forming the so-called up fragment, can also be eliminated through rewriting, in a purely local way, as described in Chapter 3. The interpretation of these rules is an open question, but a further study of the rewrite system corresponding to symmetric normalisation might lead to the understanding necessary to make good use of this refined reduction behaviour.

On the side of the proof-search-as-computation paradigm, the systems for linear logic defined in Chapter 7, and in particular the focused system, follow the same principles of transferring the refinements of the logical level into the computational interpretations. There are two benefits to the deep inference approach to focusing and related normal forms, that are worth exploring further. First, in this setting, the essential connection between polarities and focusing are made clear, through the idea that some given proof is focused only if it respects polarities, in a certain strong sense, yet to be compared to the notion of analiticity. Then, this methodology also allows for a refinement of the computation steps described by focusing phases, that are considered as atomic events, and a recomposition of these steps in ways which cannot be achieved in the sequent calculus. The use of these theoretical results in practical implementations of logic programming engines might provide significant improvements in the expressivity of the language offered to the user. Finally, notice that the correspondence described in Chapter 8 establishes a new bridge between functional and logic computation, and could represent only the first step in a larger correspondence, involving more powerful proof systems yielding more expressive languages, such as various pattern calculi.

Bibliography

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[ACCL90] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. In *POPL'90*, pages 31–46, 1990.

[AF05] Sandra Alves and Mário Florido. Weak linearization of the lambda calculus. *Theoretical Computer Science*, 342(1):79–103, 2005.

[AK10] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In A. Dawar and H. Veith, editors, *CSL'10*, volume 6247 of *LNCS*, pages 381–395, 2010.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[AR99] Michele Abrusci and Paul Ruet. Non-commutative logic I: the multiplicative fragment. *Annals of Pure and Applied Logic*, 101(1):29–64, 1999.

[Asp98] Andrea Asperti. Optimal reduction of functional expressions. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *PLILP'98*, volume 1490 of *LNCS*, pages 427–428, 1998.

[Avr96] Arnon Avron. The method of hypersequents in the proof theory of propositional non-classical logics. In *Logic: from foundations to applications: European logic colloquium*, pages 1–32. Clarendon, 1996.

[Bar84] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.

[BCL99] Gérard Boudol, Pierre-Louis Curien, and Carolina Lavatelli. A semantics for λ-calculi with resources. *Mathematical Structures in Computer Science*, 9(4):437–482, 1999.

[Bel82] Nuel Belnap. Display logic. *Journal of Philosophical Logic*, 11:375–417, 1982.

[BG96] Paola Bruscoli and Alessio Guglielmi. A linear logic view of Gamma style computations as proof searches. In J-M. Andreoli, C. Hankin, and D. Le Métayer, editors, *Coordination Programming: Mechanisms, Models and Semantics*. Imperial College Press, 1996.

[BG00] Henk Barendregt and Silvia Ghilezan. Lambda-terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10(1):121–134, 2000.

[BG03] Paola Bruscoli and Alessio Guglielmi. A tutorial on proof theoretic foundations of logic programming. In Catuscia Palamidessi, editor, *ICLP'03*, volume 2916 of *LNCS*, pages 109–127, 2003.

[BH34] Paul Bernays and David Hilbert. *Grundlagen der Mathematik, I*. 1934.

[BM08] Kai Brünnler and Richard McKinley. An algorithmic interpretation of a deep inference system. In I. Cervesato, H. Veith, and A. Voronkov, editors, *LPAR'08*, volume 5330 of *LNCS*, pages 482–496, 2008.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[BNS10] Taus Brock-Nannestad and Carsten Schürmann. Focused natural deduction. In C. Fermüller and A. Voronkov, editors, *LPAR'10*, volume 6397 of *LNCS*, pages 157–171, 2010.

[Bou93] Gérard Boudol. The λ-calculus with multiplicities (abstract). In E. Best, editor, *CONCUR'93*, volume 715 of *LNCS*, pages 1–6, 1993.

[BR95] R. Bloo and K. Rose. Preservation of strong normalisation in named λ-calculi with explicit substitution and garbage collection. In *CSN'95*, pages 62–72, 1995.

[Bru02] Paola Bruscoli. A purely logical account of sequentiality in proof search. In P. J. Stuckey, editor, *ICLP'02*, volume 2401 of *LNCS*, pages 302–316, 2002.

[Brü03] Kai Brünnler. *Deep Inference and Symmetry in Classical Proofs*. PhD thesis, Technische Universität Dresden, September 2003.

[Brü06a] Kai Brünnler. Cut elimination inside a deep inference system for classical predicate logic. *Studia Logica*, 82(1):51–71, 2006.

[Brü06b] Kai Brünnler. Deep inference and its normal form of derivations. In A. Beckmann, U. Berger, B. Löwe, and J. Tucker, editors, *CiE 2006*, volume 3988 of *LNCS*, pages 65–74, 2006.

[Brü06c] Kai Brünnler. Locality for classical logic. *Notre Dame Journal of Formal Logic*, 47:557–580, 2006.

[Brü10] Kai Brünnler. *Nested Sequents*. Habilitation thesis, Universität Bern, 2010.

[BS94] Gianluigi Bellin and Philip Scott. On the pi-calculus and linear logic. *Theoretical Computer Science*, 135:11–65, 1994.

[Bus91] Samuel Buss. The undecidability of k-provability. *Annals of Pure and Applied Logic*, 53:72–102, 1991.

[Cha08] Kaustuv Chaudhuri. Focusing strategies in the sequent calculus of synthetic connectives. In I. Cervesato, H. Veith, and A. Voronkov, editors, *LPAR'08*, volume 5330 of *LNCS*, pages 467–481, 2008.

[CMS08] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In G. Ausiello, J. Karhumäki, G. Mauri, and L. Ong, editors, *Fifth IFIP International Conference on Theoretical Computer Science*, volume 273, pages 383–396, September 2008.

[CR36] Alonzo Church and J. B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936.

[Cur34] Haskell Curry. Functionality in combinatorial logic. In *Proceedings of National Academy of Sciences*, volume 20, pages 584–590, 1934.

[Cur52] Haskell Curry. The permutability of rules in the classical inferential calculus. *Journal of Symbolic Logic*, 17(4):245–248, 1952.

[dB72] Nicolaas de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[DP99] Roy Dyckhoff and Luís Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212(1–2):141–155, 1999.

[Dyc92] Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992.

[ER03] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1–3):1–41, 2003.

[FMS05] M. Fernández, I. Mackie, and F-R. Sinot. Lambda-calculus with director strings. *Applicable Algebra in Engineering, Communication and Computing*, 15(6):393–437, 2005.

[Gal93] Jean Gallier. Constructive logics part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

[Gan80] Robin Gandy. Proofs of strong normalisation. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[Gen34] Gerhard Gentzen. Untersuchungen über das logische Schließen, I. *Math. Zeitschrift*, 39:176–210, 1934.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen, II. *Math. Zeitschrift*, 39:405–431, 1935.

[GGP10] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In C. Lynch, editor, *RTA'10*, volume 6 of *LIPIcs*, pages 135–150, 2010.

[GGS11] Alessio Guglielmi, Tom Gundersen, and Lutz Straßburger. Breaking paths in atomic flows for classical logic. In *LICS'10*, pages 284–293, 2011.

[Gir71] Jean-Yves Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Second Scandinavian Logic Symposium*, pages 63–92. North-Holland, Amsterdam, 1971.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir91] Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[Gir96] Jean-Yves Girard. Proof-nets : the parallel syntax for proof-theory. In A. Ursini and P. Agliano, editors, *Logic and Algebra*. M. Dekker, New York, 1996.

[Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[GS02] Alessio Guglielmi and Lutz Straßburger. A non-commutative extension of MELL. In M. Baaz and A. Voronkov, editors, *LPAR'02*, volume 2514 of *Lecture Notes in Artificial Intelligence*, pages 231–246, 2002.

[GS09] Alessio Guglielmi and Lutz Straßburger. A system of interaction and structure V: The exponentials and splitting, 2009. To appear in Mathematical Structures in Computer Science.

[GS11a] Alessio Guglielmi and Lutz Straßburger. A system of interaction and structure IV: The exponentials and decomposition. *ACM Transactions on Computational Logic*, 12(4), 2011.

[GS11b] Alessio Guglielmi and Lutz Straßburger. A system of interaction and structure V: The exponentials and splitting. *Mathematical Structures in Computer Science*, 21(3):563–584, 2011.

[Gug99] Alessio Guglielmi. A calculus of order and interaction. Technical Report WV-99-04, Technische Universität Dresden, 1999.

[Gug07] Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, 2007.

[Her94] Hugo Herbelin. A λ-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *CSL'94*, volume 933 of *LNCS*, pages 61–75, 1994.

[HM94] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[How80] William Howard. The Formulae-As-Types Notion Of Construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[Hug10] Dominic Hughes. A classical sequent calculus free of structural rules. *Annals of Pure and Applied Logic*, 161:1244–1253, July 2010.

[JK09] Barry Jay and Delia Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.

[JM00] F. Joachimski and R. Matthes. Standardization and confluence for a lambda calculus with generalized applications. In L. Bachmair, editor, *RTA'00*, volume 1833 of *LNCS*, pages 141–155, 2000.

[Kah06] O. Kahramanoğulları. *Nondeterminism and Language Design in Deep Inference*. PhD thesis, Technische Universität Dresden, December 2006.

[Kas94] Ryo Kashima. Cut-free sequent calculi for some tense logics. *Studia Logica*, 53(1):119–136, 1994.

[Kes07] D. Kesner. The theory of calculi with explicit substitutions revisited. In J. Duparc and T. A. Henzinger, editors, *CSL'07*, volume 4646 of *LNCS*, pages 238–252, 2007.

[KL05] D. Kesner and S. Lengrand. Extending the explicit substitution paradigm. In J. Giesl, editor, *RTA'05*, volume 3467 of *LNCS*, pages 407–422, 2005.

[KL07] D. Kesner and S. Lengrand. Resource operators for the λ-calculus. *Information and Computation*, 205(4):419–473, 2007.

[Kle45] Stephen Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945.

[Kle52] Stephen Kleene. Permutabilities of inferences in gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 10:1–26, 1952.

[Klo80] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht Universiteit, 1980.

[KR11] Delia Kesner and Fabien Renaud. A prismoid framework for languages with resources. *Theoretical Computer Science*, 412(37):4867–4892, 2011.

[Kri63] Saul Kripke. Semantical analysis of intuitionistic logic I. In J. Crossley and M. Dummet, editors, *Formal Systems and Recursive Functions*, pages 92–130. North Holland, 1963.

[KS88] Richard Kennaway and Ronan Sleep. Director strings as combinators. *ACM Transactions on Programming Languages and Systems*, 10(4):602–626, 1988.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1–2):163–180, 2004.

[Lau02] Olivier Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.

[Lau04] Olivier Laurent. A proof of the focalization property of linear logic. May 2004.

[Les94] Pierre Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions. In *POPL'94*, pages 60–69, 1994.

[LM07] C. Liang and D. Miller. Focusing and polarization in intuitionistic logic. In J. Duparc and T. A. Henzinger, editors, *CSL'07*, volume 4646 of *LNCS*, pages 451–465, 2007.

[LM09] C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

[McC60] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, 3(4):184–195, 1960.

[Mel95] P-A. Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *TLCA'95*, volume 902 of *LNCS*, pages 328–334, 1995.

[Mel10] Paul-André Melliès. Resource modalities in tensor logic. *Annals of Pure and Applied Logic*, 161(5):632–653, 2010.

[Mil99] R. Milner. *Communicating and Mobile Systems : The π-calculus*. Cambridge University Press, 1999.

[MNPS91] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Mog89] E. Moggi. Computational λ-calculus and monads. In *LICS'89*, pages 14–23, 1989.

[MS07] D. Miller and A. Saurin. From proofs to focused proofs : a modular proof of focalization in linear logic. In J. Duparc and T. A. Henzinger, editors, *CSL'07*, volume 4646 of *LNCS*, pages 405–419, 2007.

[MW88] David Maier and David Warren. *Computing With Logic: Logic Programming With Prolog*. Addison-Wesley, 1988.

[New42] M. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[NvP01] Sara Negri and Jan von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

[Pol04] Emmanuel Polonowski. *Substitutions Explicites, Logique et Normalisation*. PhD thesis, Université Paris-Diderot — Paris VII, 2004.

[Pra65] Dag Prawitz. *Natural Deduction: a proof-theoretical study*. PhD thesis, Almqvist & Wiksell, 1965.

[PT98] Benjamin Pierce and David Turner. Local type inference. In D. MacQueen and L. Cardelli, editors, *POPL'98*, pages 252–265, 1998.

[Pym02] David Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer, 2002.

[Ren11] Fabien Renaud. *Les Ressources Explicites vues par la Théorie de la Réécriture*. Thèse de doctorat, Université Paris-Diderot, 2011.

[Ret97] Christian Retoré. Pomset logic: A non-commutative extension of classical linear logic. In P. de Groote, editor, *TLCA'97*, volume 1210 of *LNCS*, pages 300–318, 1997.

[Rey74] John Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Symposium on Programming*, volume 19 of *LNCS*, pages 408–423, 1974.

[Rey98] John Reynolds. *Theories of programming languages*. Cambridge University Press, 1998.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Rov11] Luca Roversi. Linear λ-calculus and deep inference. In L. Ong, editor, *TLCA'11*, volume 6690 of *LNCS*, pages 184–197, 2011.

[San93] D. Sangiorgi. From π-calculus to higher-order π-calculus - and back. In M-C. Gaudel and J-P. Jouannaud, editors, *TAPSOFT'93*, volume 668 of *LNCS*, pages 151–166, 1993.

[San09] José Espírito Santo. The λ-calculus and the unity of structural proof theory. *Theory of Computing Systems*, 45(4):963–994, 2009.

[Sch60] Kurt Schütte. *Beweistheorie*. Springer, 1960.

[Sel07] P. Selinger. Lecture notes on the λ-calculus. 2007.

[SH11] Peter Schroeder-Heister. Implications-as-rules vs. implications-as-links: an alternative implication-left schema for the sequent calculus. *Journal of Philosophical Logic*, 40:95–101, 2011.

[Str03a] Lutz Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, Technische Universität Dresden, July 2003.

[Str03b] Lutz Straßburger. MELL in the calculus of structures. *Theoretical Computer Science*, 309(1–3):213–285, 2003.

[Str07] Lutz Straßburger. A characterisation of medial as rewriting rule. In F. Baader, editor, *RTA'07*, volume 4533 of *LNCS*, pages 344–358, 2007.

[SU06] M. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism, Volume 149, Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006.

[Tiu06a] Alwen Tiu. A local system for intuitionistic logic. In M. Hermann and A. Voronkov, editors, *LPAR'06*, volume 4246 of *LNCS*, pages 242–256, 2006.

[Tiu06b] Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2:4):1–24, 2006.

[TM04] Alwen Tiu and Dale Miller. A proof search specification of the π-calculus. In *Third Workshop on the Foundations of Global Ubiquitous Computing*, volume 138 of *ENTCS*, pages 79–101, 2004.

[Tro03] Anne Troelstra. Constructivism and proof theory, 2003.

[TS96] Anne Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.

[Vig00] Luca Viganò. *Labelled Non-classical Logics*. Kluwer, 2000.

[Yet90] David Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55(1):41–64, 1990.