YALLA: Yet Another deep embedding of Linear Logic in Rocq

Olivier Laurent

Laboratoire de l'Informatique du Parallélisme, CNRS – ENS de Lyon – Université Claude Bernard Lyon 1, LIP – UMR 5668, 69342 Lyon cedex 07, France.

Corresponding author(s). E-mail(s): Olivier.Laurent@ens-lyon.fr;

Abstract

We define a parameterized system for propositional Linear Logic which allows for a formalization in `Rocq` which can be instantiated to the specific needs of the user. Parameters control the use of the exchange and mix rules in particular. The induced `Rocq` library provides basic key properties of the system (and of some variants) such as cut admissibility, focusing, etc.

Keywords: linear logic, proof assistant, cut elimination, the Rocq prover, formalization

1 Introduction

This work took its original inspiration from a talk presented at LSFA 2016 (Chaudhuri et al. 2017) where propositional linear logic was formalized in Abella with sequents/contexts represented as finite multisets. Such formalizations with *sequents as multisets* impacts the representation of the computational content of proofs (as defined by the Curry-Howard-Lambek correspondence) by identifying proofs with different computational behaviours.

[0]This version of the article has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/s10817-026-09755-y

Let us consider the following first two proofs (where indices are only used to distinguish occurrences of the same propositional variable):

$$\dfrac{\dfrac{\dfrac{}{X_1 \vdash X_0}}{X_1, X_2 \vdash X_0}}{\dfrac{X_1 \vdash X_2 \to X_0}{\vdash X_1 \to X_2 \to X_0}} \qquad \dfrac{\dfrac{\dfrac{}{X_2 \vdash X_0}}{X_1, X_2 \vdash X_0}}{\dfrac{X_1 \vdash X_2 \to X_0}{\vdash X_1 \to X_2 \to X_0}} \qquad \dfrac{\dfrac{\dfrac{}{X \vdash X}}{X, X \vdash X}}{\dfrac{X \vdash X \to X}{\vdash X \to X \to X}}$$

Through the Curry-Howard isomorphism, they are associated with the two simply-typed λ-terms $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$ (the standard Church encodings of the two Booleans true and false) with type $X \to X \to X$. However if the left-hand side of the sequent is a multiset then there is no possible distinction between $X_1, X_2 \vdash X_0$ and $X_2, X_1 \vdash X_0$, so that without additional informations, there is no way to know if the top-most left variable is X_1 or X_2: we get the third proof above and we are loosing the possibility to follow occurrences. This means the two proofs are identified and this makes their computational interpretations as Booleans collapse. In the setting of linear logic, the same phenomenon occurs with the, so called, *multiplicative Booleans* for example:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X_1 \vdash X_3} \quad \dfrac{}{X_2 \vdash X_4}}{X_1, X_2 \vdash X_3 \otimes X_4}}{X_1 \vdash X_2 \multimap X_3 \otimes X_4}}{\vdash X_1 \multimap X_2 \multimap X_3 \otimes X_4} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{X_2 \vdash X_3} \quad \dfrac{}{X_1 \vdash X_4}}{X_1, X_2 \vdash X_3 \otimes X_4}}{X_1 \vdash X_2 \multimap X_3 \otimes X_4}}{\vdash X_1 \multimap X_2 \multimap X_3 \otimes X_4}$$

with associated linear λ-terms $\lambda x.\lambda y.\langle x, y\rangle$ and $\lambda x.\lambda y.\langle y, x\rangle$ of type $X \multimap X \multimap X \otimes X$.

A solution to this problem is to representent sequents and contexts as *lists*. In many cases this requires the introduction of an explicit exchange rule and forces to make explicit manipulations on permutations (which act on lists). Our starting point was to see if replaying (Chaudhuri et al. 2017) in `Coq`[1] with such an explicit exchange rule would be manageable or not.

Following standard approaches (see Section 7 for example) in formalizing logical systems in `Rocq` (formerly `Coq`), we started (`Yalla` version 1) with a representation of the one-sided sequent-calculus linear logic proofs with type: `list formula -> Prop`. However because of the structure/properties of the logical framework underlying `Rocq`, using `Prop` is yet another way to lose the computational content of proofs since it is not possible to extract for example a non-trivial Boolean from an object in `Prop`. Here comes `Yalla` version 2 with `Prop` replaced with `Type`.

To sum up, starting from the goal of providing an encoding of linear logic (Girard 1987) preserving the computational content of proofs, the key specific ingredients on which our present work is built are an *explicit exchange rule* acting on *sequents as lists*, and sequent-calculus proofs defined as *first-class objects in* `Type` (rather than a provability predicate in `Prop`), *i.e.* using the type: `list formula -> Type`.

[1] The choice of `Coq` (rather than `Abella` or another proof assistant) was mostly based on the size of the community and on the expertise of the author.

Then came the idea of not just defining yet another formalization of linear logic in Rocq (see Section 7 for an already rather long list of Rocq developments) but to go towards a reusable library which would provide the necessary standard results about linear logic to allow users to formalize their own new contributions without replaying the basics (an heavy entrance cost to pay otherwise). The goal of being generic enough required dealing with as many variants as possible through the introduction of various parameters to fulfil the needs of users: different expressive powers of the exchange rule related with non-commutative logics, introduction of the mix rules, etc. Many results and linear-logic-related systems have been incorporated and in particular almost all the logics and results presented in (Laurent 2018) are included in the Yalla library.

We will focus on Yalla version 2.0 (last minor release so far being 2.0.7 for Rocq 9.0–9.2):

https://github.com/olaure01/yalla/tree/v2.0._

A major restriction of the current version is that only *propositional* linear logic is considered. Quantifiers or type fixpoints are currently out of the scope. This is justified by technical difficulties related with the formalization of binders, but also because most novelties introduced by linear logic act at the level of propositional connectives.

Structure of the Paper.

In the next two sections (Sections 1.1 and 1.2), we briefly introduce some notations and terminology about sequent-style proofs. The paper then contains two very different parts.

The first one (Sections 2 to 5) describes, in logical terms, the considered systems and results in a way which is guided by formalization but readable without any particular knowledge about proof-assistants. Many results here are standard or folklore but we give them to show how they can be proved (and sometimes generalized), how they relate, and how they are formalized since the proofs written in this paper are those used in the Rocq development. More advanced results formalized in Yalla come from (Laurent 2018) and we will mostly refer to this paper for them. We try to give all ingredients, and to stress key points leading to choices in the formalization. However since proofs are given with all details in the Rocq files, we sometimes give just a sketch in the paper, or only a few key cases. Some references pointing directly to the Rocq code are provided in the form [file.v#statement] which leads to the statement statement in file https://github.com/olaure01/yalla/tree/paper_jar/yalla/file.v

The second part (Sections 6 and 7) focuses on specific technical details of the formalization in Rocq. An overview of the content of the library can be seen on Figure 1 through the dependency graph of the Rocq files. Files in blue are those dealing with classical linear logic. Files in red are those dealing with intuitionistic linear logic. Files in magenta provide relations between classical and intuitionistic linear logics. Files in cyan are coming from the microyalla kernel. Files in lighter blue are template files deriving instances of linear-logic-related systems from the main parameterized system. Files in green lead to the proof of the focusing property. Finally those in black contain more technical or specific material.

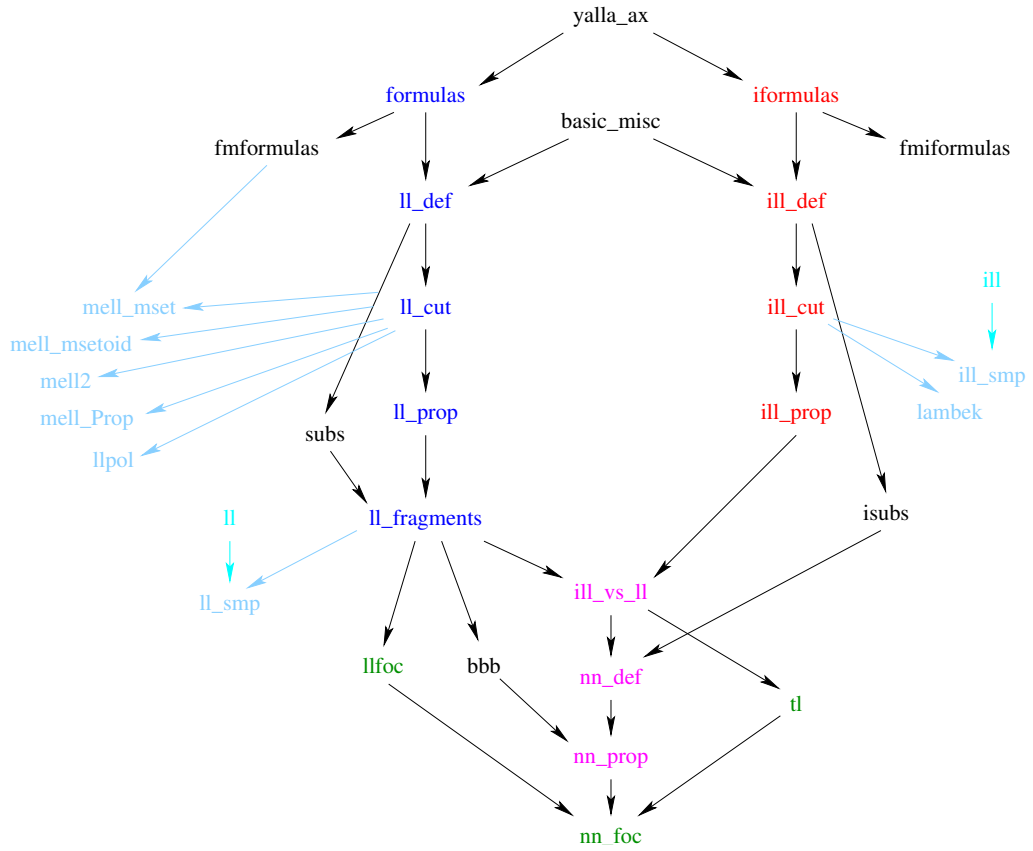


Fig. 1 Dependency graph of `Yalla` `Rocq` files.

How to Read the Paper.

As a Linear Logician. This paper is written as a journey through key core results about linear logic, and how they depend on some variations on the logic. Most results are well known by anyone with basic knowledge about linear logic, some are folklore, and some are more original. We hope that even linear logic experts could learn at least one new thing in this paper.

The main specificity of the paper lies behind the way these results are presented: everything is extracted from the `Rocq` formalization, even if we made the choice to include almost no `Rocq` code for readability. Results are presented in standard mathematical language and pointers are provided to the source code implementing them.

Our main objective is to point out the technical details often hidden in pen-and-paper approaches to linear logic. On the one hand, this shows that these details may lead to logical questions deserving some attention. On the other hand, it should provide to linear logicians the key ingredients to have in mind when trying to move to proof assistants for dealing with linear logic. Our hope is that, being aware of these

particularities and thanks to the results already formalized in the `Yalla` library, linear logicians should not be afraid of trying to formalize their new results on machine in the future.

Reading is recommanded from the beginning of the paper and could stop at the end of Section 5. Driven by curiosity, the reader may click on some `links to Rocq data` to see formalization details. Appendices C and D provide two concrete examples of use of the `Yalla` library with comments. Then looking at one of the template files, such as `mell2.v`, should enlighten how other variants of linear logic can be represented in `Rocq` and a how to bridge them with the `Yalla` library in order to take benefits from the results it contains. Additional useful informations about the formalization can be found in Section 6, in particular Section 6.3 about exchange and Section 6.4 about the parameters.

As a Proof-Assistant User. Even if the presented results are often basic ones from the theory of linear logic, and even if all necessary ingredients are presented, this is *not* designed as a proper introduction to linear logic. Too many discussions deal here with very particular (tedious?) details about the theory coming from formalization and which should be ignored in a first approach to linear logic. However these details are important from a formalization point of view.

For a reader more interested in the formalization aspects, the design choices and their impacts, we recommend to read from the beginning but also to click on each `link to Rocq data` (note the `Rocq` code associated with the key Figure 2 page 8 is reproduced in Figure 8 page 36) and to stop before Section 2.2.1. Then jumping to Section 6 should provide more informations about design choices. After that, Section 2.2.2 presents key results formalized in `Yalla` (cut admissbility in particular). Finally most other points developed about linear logic can be read independently, depending on the knowledge and interests of the reader (still clicking on `Rocq links` to watch pieces of formalization in their natural habitat).

1.1 Notations for Lists and Permutations

We will use the following notations about lists: $[a]$ is the singleton list containing a, $\overline{l}$ is the reverse of the list l and $l_1 +\!\!+ l_2$ is the concatenation of the two lists l_1 and l_2.

$\mathfrak{S}$ is the set of all finite permutations. $\mathfrak{O}$ is the set of all cyclic permutations (a.k.a. circular shifts). Permutations in $\mathfrak{O}$ are those which only rotate elements: $(a, b, c, d, e) \mapsto (c, d, e, a, b)$ for example. Two lists l_1 and l_2 are related through a cyclic permutation if and only if there exist l' and l'' such that $l_1 = l' +\!\!+ l''$ and $l_2 = l'' +\!\!+ l'$.

1.2 Rules

We will consider various sequent-based deduction systems. Given a notion of formula, a sequent is a structure containing formulas (for example lists, pairs of lists, etc.). A proof is then a tree whose nodes are specified by the rules of the system and edges are labeled by sequents. We distinguish between *proofs* where leaves must be given by nullary rules and *open proofs* which are "partial" proofs where any sequent is allowed to be a leaf. An n-ary rule R is then a description of a valid node of a proof allowing to deduce a sequent S from n previously derived sequents $S_1, \ldots, S_n$. It has the shape:

$$\frac{S_1 \qquad \cdots \qquad S_n}{S}\, R$$

$S_1, \ldots, S_n$ are the premises of R and S is its conclusion.

Admissibility and Derivability.

Since we are going to consider variants of linear logic, it is useful to be able to compare the expressiveness of different sets of rules. Given a set of rules $\mathcal{R}$, the rule R is *admissible* in $\mathcal{R}$ if, whenever $S_1, \ldots, S_n$ are provable using the rules of $\mathcal{R}$, then S is also provable with the rules of $\mathcal{R}$. One can then consider the system $\mathcal{R} \cup \{R\}$ which proves the same sequents as $\mathcal{R}$. To make the situation clear, we use a specific notation for admissible rules:

$$\frac{S_1 \quad \cdots \quad S_n}{S}\, R$$

when a currently used system $\mathcal{R}$ is clear from the context.

The rule R is *derivable* from the set $\mathcal{R}$ if there is an *open* proof in $\mathcal{R}$ with $S_1, \ldots, S_n$ as premises and S as conclusion. For derivable rules, we use the notation:

$$\frac{\frac{S_1 \qquad \cdots \qquad S_n}{}}{S}\, R$$

Note that if R is derivable from $\mathcal{R}$, then it is admissible in $\mathcal{R}$ (and the two notions coincide for nullary rules).

Let us now consider two additional sets of rules $\mathcal{S}_1$ and $\mathcal{S}_2$. We say that $\mathcal{S}_1$ and $\mathcal{S}_2$ are *inter-derivable* over $\mathcal{R}$ if each rule in $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) is derivable in $\mathcal{R} \cup \mathcal{S}_2$ (resp. $\mathcal{R} \cup \mathcal{S}_1$). In practice, inter-derivable sets of rules provide alternative ways of extending the system $\mathcal{R}$ which both lead to the same provable sequents.

Invertibility.

A rule $\dfrac{S_1 \qquad \cdots \qquad S_n}{S}$ is *invertible* (or *reversible*) if all the "converse" unary rules $\dfrac{S}{S_i}$ are admissible. From this definition, a nullary rule is always invertible.

2 Classical Linear Logic

2.1 Formulas and Sequents

Given a set `Atom` (whose elements are denoted X, Y, etc.) of propositional variables, *formulas* of propositional classical linear logic are defined by the grammar:

$$A ::= X \mid \tilde{X} \mid 1 \mid \bot \mid A \otimes A \mid A \parr A \mid 0 \mid \top \mid A \oplus A \mid A \,\&\, A \mid !A \mid ?A$$

Formulas X and $\tilde{X}$ are called *atomic*. Connectives $\otimes$, $\parr$, 1 and $\bot$ are called *multiplicatives*. Connectives $\oplus$, $\&$, 0 and $\top$ are called *additives*. Connectives ! and ? are called *exponentials*.

The *dual* $A^\perp$ of a formula A is defined inductively by:

$$\begin{array}{rcl@{\qquad}rcl} X^\perp & := & \tilde{X} & \tilde{X}^\perp & := & X \\ 1^\perp & := & \bot & \bot^\perp & := & 1 \\ (A \otimes B)^\perp & := & B^\perp \parr A^\perp & (A \parr B)^\perp & := & B^\perp \otimes A^\perp \\ 0^\perp & := & \top & \top^\perp & := & 0 \\ (A \oplus B)^\perp & := & A^\perp \,\&\, B^\perp & (A \,\&\, B)^\perp & := & A^\perp \oplus B^\perp \\ (!A)^\perp & := & ?A^\perp & (?A)^\perp & := & !A^\perp \end{array}$$

The change of order of sub-formulas in $(A \otimes B)^\perp$ and $(A \parr B)^\perp$ is required to deal with non-commutative systems (see Section 2.2.1). In commutative cases, we have $A \otimes B \simeq B \otimes A$ and both definitions $(A \parr B)^\perp := B^\perp \otimes A^\perp$ and $(A \parr B)^\perp := A^\perp \otimes B^\perp$ are valid. However in a non-commutative setting, only the choice above is possible. The additive connectives being commutative, even in non-commutative linear logics, the order of sub-formulas does not matter for them and we keep the usual one.

Lemma 1 (Bidual [`formulas.v#bidual`]) *Let A be a formula, $(A^\perp)^\perp = A$.*

Definition 1 (Size of a formula [`formulas.v#fsize`]) The *size* $|A|$ of a formula A is the number of connectives in A with atomic formulas counting for 1 ($|X| = |\tilde{X}| = 1$).

A sequent is a *list* of formulas, denoted $\vdash \Gamma$. We consider implicit mappings of unary formula-operations on lists of formulas: $?\Gamma$ or $!\Gamma$ or $\Gamma^\perp$ means applying $?$, $!$ or $(\_)^\perp$ to each element of Γ.

In classical linear logic, we sometimes use $A \vdash B$ as a notation for $\vdash A^\perp, B$.

2.2 Proofs

Rather than a single sequent calculus system for classical linear logic, we consider a parameterized family $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$. This family of systems contains usual LL but also some standard variants. The *five parameters* are:

- $\mathcal{A}$ is a set of lists of formulas (considered as valid sequents in the system, *i.e.* axioms);
- p is a Boolean controlling the impact of the exchange rule, thus commutativity aspects of the logic;
- c is a Boolean controlling whether or not a primitive *cut* rule belongs to the system;
- m_0 and m_2 are Booleans controlling the use of the *mix* rules.

Specific values of the parameters provide concrete sequent calculus systems as instances of the family. This approach, inspired by the notion of level in (Pous 2012), allows us to factorize proofs of properties which hold for more than one value of the parameters.

$$\frac{}{\vdash \tilde{X}, X}\, ax \qquad (\Gamma \in \mathcal{A})\ \frac{}{\vdash \Gamma}\, ax_g \qquad (c)\ \frac{\vdash A^{\perp}, \Gamma \qquad \vdash A, \Delta}{\vdash \Delta, \Gamma}\, cut$$

$$(\sigma \in \mathfrak{S}_p)\ \frac{\vdash \Gamma}{\vdash \sigma(\Gamma)}\, ex \qquad (\sigma \in \mathfrak{S})\ \frac{\vdash \Gamma, ?\Delta, \Sigma}{\vdash \Gamma, \sigma(?\Delta), \Sigma}\, ex_?$$

$$(m_0)\ \frac{}{\vdash}\, mix_0 \qquad (m_2)\ \frac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Delta, \Gamma}\, mix$$

$$\frac{}{\vdash 1}\, 1 \qquad \frac{\vdash \Gamma}{\vdash \perp, \Gamma}\, \perp \qquad \frac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Delta, \Gamma}\, \otimes \qquad \frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}\, \parr$$

$$\frac{}{\vdash \top, \Gamma}\, \top \qquad \frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma}\, \oplus_1 \qquad \frac{\vdash A, \Gamma}{\vdash B \oplus A, \Gamma}\, \oplus_2 \qquad \frac{\vdash A, \Gamma \qquad \vdash B, \Gamma}{\vdash A \,\&\, B, \Gamma}\, \&$$

$$\frac{\vdash A, ?\Gamma}{\vdash !A, ?\Gamma}\, ! \qquad \frac{\vdash A, \Gamma}{\vdash ?A, \Gamma}\, ?d \qquad \frac{\vdash \Gamma}{\vdash ?A, \Gamma}\, ?w \qquad \frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}\, ?c$$

Fig. 2 The $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ family of systems.

Before presenting the rules of the system, let us define the parameterized set $\mathfrak{S}_p$:

$$\mathfrak{S}_p = \begin{cases} \mathfrak{S} & \text{if } p = \texttt{true} \\ \mathfrak{O} & \text{if } p = \texttt{false} \end{cases}$$

The *rules of* $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ are given in Figure 2. Some of them depend on the values of the five parameters. The constraints related with the parameters are written on the left of the rules. For example, the constraint (c) on the (cut) rule means that the rule can be applied only in the instances of the shape $\mathsf{LL}(\mathcal{A}, p, \texttt{true}, m_0, m_2)$ where $c = \texttt{true}$, otherwise the rule is not considered to be part of the instance of the system.

The standard definition of the sequent calculus of propositional classical linear logic corresponds to $\mathsf{LL}(\emptyset, \texttt{true}, \texttt{true}, \texttt{false}, \texttt{false})$ (*i.e.* $\mathcal{A} = \emptyset$ (no extra axioms), $p = \texttt{true}$ (full exchange), $c = \texttt{true}$ (explicit cut rule), $m_0 = \texttt{false}$ (no mix_0 rule) and $m_2 = \texttt{false}$ (no *mix* rule)) which, thanks to cut-admissibility (Theorem 1), has the same provability as the variant with $c = \texttt{false}$.

Let us now comment on the impact and use of the parameters:

- The generalized axiom rule (ax_g) which uses the parameter $\mathcal{A}$ is included for two main reasons. First this is the way we are going to represent *open proofs*, *i.e.* partial proofs where some hypotheses remain unproved (as used in the notion of derivable rule, see Section 1.2). The allowed open hypotheses have to be put in $\mathcal{A}$ (if $\mathcal{A} = \emptyset$, the rule cannot be used and the proof is closed).
  The second reason is to be able to consider linear logic enriched with theories (see for example (Lincoln et al. 1992; Baillot and Das 2016)). Elements of $\mathcal{A}$ are then the axioms of the theory. Similary this allows to inject a graph or a category in

the axiom rules as used for building free categories (see (Lambek and Scott 1988, Part I, Sections 1–4)). Elements of $\mathcal{A}$ are then the types of arrows.

- In the presence of the (ax_g) rule, it becomes necessary to integrate the (cut) rule in the system: although the (cut) rule will be proved admissible when $\mathcal{A} = \emptyset$ (Theorem 1), it is not the case for arbitrary $\mathcal{A}$. As a consequence, for a non trivial $\mathcal{A}$, the instances with $c = \texttt{true}$ and with $c = \texttt{false}$ may have different expressive power.
- The two *mix* rules (mix_0 and mix) are often considered in variants of linear logic. They do not have a very strong impact on the theory of the system: most of the proofs of the meta-properties are the same for "core" linear logic and for the versions extended with *mix* rules. The parameters m_0 and m_2 then allow to uniformly consider the four induced systems.
- The parameter p controls the exchange rule and thus the expressiveness of the system in terms of commutativity. We discuss this aspect in more details in the next section since there are multiple variants and possible design choices. The order of the formulas and contexts in the conclusion $(A \otimes B, \Delta, \Gamma)$ of the $(\otimes)$ rule is important with respect to non-commutativity constraints (see Section 2.2.1): this order is required for the non-commutative (cyclic) case and does not matter in the commutative one. In the conclusion of the (cut) rule (and similarly for (mix)), Γ, Δ instead of Δ, Γ would make no real difference:

$$(c)\ \frac{\vdash A^\perp, \Gamma \qquad \vdash A, \Delta}{\vdash \Gamma, \Delta} \qquad\qquad (m_2)\ \frac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Gamma, \Delta}$$

since these rules are immediately derivable from the presented ones by using $A^{\perp\perp} = A$ (Lemma 1). We prefer Δ, Γ for consistency with the choice in the $(\otimes)$ rule.

2.2.1 Exchange Rules and Non-Commutative Logic

Regarding commutativity, we are interested in two key variants of linear logic: the standard commutative version and the cyclic version (Yetter 1990). In both cases, there are many equivalent ways of formalizing the exchange rule. While the exchange rule is necessary in the logic (see also discussion in Section 6.3), this is not the main rule we want to focus on. For this reason we are happy to be able to limit its use (*i.e.* to replace sequences of exchange rules by a more compact single application). This is a way to get more "canonical" objects. For this reason we prefer presentations of exchange rules which admit closure under composition, meaning that two consecutive applications of the rule can be replaced by just one. While explicit exchange rules are important in a Curry-Howard-Lambek perspective (see Section 6.3), such a collapse of consecutive exchange rules has impact neither on the computational meaning of proofs nor on denotational semantics.

Cyclic Exchange Rule.

Let us first discuss the cyclic exchange rule:

$$\frac{\vdash \Delta, \Gamma}{\vdash \Gamma, \Delta}\ exc$$

It is an instance of both variants $p = \texttt{true}$ and $p = \texttt{false}$ of the (ex) rule since $\mathfrak{O} \subseteq \mathfrak{S}_p$ (independently of the value of p). Moreover it is closed under composition:

$$\frac{\dfrac{\vdash \Sigma, \Gamma, \Delta}{\vdash \Delta, \Sigma, \Gamma}\ exc}{\vdash \Gamma, \Delta, \Sigma}\ exc \qquad \mapsto \qquad \frac{\vdash \Sigma, \Gamma, \Delta}{\vdash \Gamma, \Delta, \Sigma}\ exc$$

There are alternative (more atomic) formulations of the cyclic exchange rule, but they happen to be equivalent with the (exc) rule and not closed under composition:

Lemma 2 (Cyclic Exchange Rules [addendum/exchange.v#*exc*]) *The following (singleton) sets of rules are inter-derivable over* $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2) \setminus \{ex, ex_?\}$*:*

$$\left\{\frac{\vdash \Delta, \Gamma}{\vdash \Gamma, \Delta}\ exc\right\} \qquad \left\{\frac{\vdash \Gamma, A}{\vdash A, \Gamma}\ exc_1^l\right\} \qquad \left\{\frac{\vdash A, \Gamma}{\vdash \Gamma, A}\ exc_1^r\right\}$$

where, $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ *being seen as a set of rules,* $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2) \setminus \{ex, ex_?\}$ *is* $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ *in which we remove the exchange rules (ex) and (ex*$_?$*).*

In the presence of cut (*i.e.* if the cut rule is admissible), the cyclic exchange rule can hardly be avoided. This is the reason why we are interested in cyclic permutations. Indeed, with cut, the admissibility of the (exc) rule happens to be equivalent to the admissibility of the extended axiom rule:

$$\frac{}{\vdash A, A^\perp}\ ax_e$$

Lemma 3 (Axiom Expansion [ll_def.v#ax_exp]) *Let* A *be a formula, the extended axiom rule* (ax_e) *is derivable.*

Proof By induction on the formula A. For example:

$$\dfrac{\dfrac{\dfrac{\dfrac{IH \qquad IH}{\vdash A, A^\perp \qquad \vdash B, B^\perp}\otimes}{\vdash A \otimes B, B^\perp, A^\perp}\ exc}{\vdash B^\perp, A^\perp, A \otimes B}\ \parr}{\vdash B^\perp \parr A^\perp, A \otimes B}\ exc}{\vdash A \otimes B, B^\perp \parr A^\perp}$$

where we see that cyclic exchange is enough as soon as we have the appropriate definition of the dual of a formula: $(A \otimes B)^\perp = B^\perp \parr A^\perp$. This explains why the definition of $(A \otimes B)^\perp$ is crucial in the cyclic case. □

Remark. Note Lemma 3 also proves $\vdash A^\perp, A$ since $(A^\perp)^\perp = A$ (Lemma 1).

In the opposite direction, as soon as (ax_e) and (cut) are admissible, (exc) is admissible as well:

$$\dfrac{\dfrac{}{\vdash A^\perp, A}\,ax_e \qquad \vdash A, \Gamma}{\vdash \Gamma, A}\,cut$$

(see [addendum/ex_from_cut.v#exc_r_cut]).

One could argue that the relation between cut admissibility and cyclic exchange is due to our particular representation of the cut rule which includes some cyclic aspect. But this relation happens to be more intrinsic as alternative presentations of rules trying to avoid cyclic exchange lead to admissibility of cyclic exchange anyway (Abrusci and Maringelli 1998).

We thus want (exc) to be a valid rule in all the variants we consider, and this is the case through instances of the (ex) rule. Then we take benefits from (exc) to restrict the other rules to the particular case where active formulas always stand at the start of sequents. Cyclic permutations allow us to generalize such rules with an additional context on the left of these active formulas. For example:

$$\dfrac{\vdash \Gamma', A, B, \Gamma}{\vdash \Gamma', A \parr B, \Gamma} \qquad := \qquad \dfrac{\dfrac{\dfrac{\vdash \Gamma', A, B, \Gamma}{\vdash A, B, \Gamma, \Gamma'}\,exc}{\vdash A \parr B, \Gamma, \Gamma'}\,\parr}{\vdash \Gamma', A \parr B, \Gamma}\,exc$$

It is slightly more tricky for multiplicative binary rules but we have for example:

$$\dfrac{\dfrac{\dfrac{\vdash \Gamma', A, \Gamma}{\vdash A, \Gamma, \Gamma'}\,exc \qquad \dfrac{\vdash \Delta', B, \Delta}{\vdash B, \Delta, \Delta'}\,exc}{\vdash A \otimes B, \Delta, \Delta', \Gamma, \Gamma'}\,\otimes}{\vdash \Gamma, \Gamma', A \otimes B, \Delta, \Delta'}\,exc \qquad (c)\ \dfrac{\dfrac{\dfrac{\vdash \Gamma', A^\perp, \Gamma}{\vdash A^\perp, \Gamma, \Gamma'}\,exc \qquad \dfrac{\vdash \Delta', A, \Delta}{\vdash A, \Delta, \Delta'}\,exc}{\vdash \Delta, \Delta', \Gamma, \Gamma'}\,cut}{\vdash \Gamma', \Delta, \Delta', \Gamma}\,exc$$

Full Exchange Rule.

Assume now that cyclic exchange holds. There are different ways of extending it to all permutations. Even if it is not necessary to get cut admissibility (Theorem 1), full exchange is important to reach the expressive power of usual *commutative* linear logic.

Lemma 4 (Exchange Rules [addendum/exchange.v#*ext*]) *The following (singleton) sets of rules are inter-derivable over* $\mathtt{LL}(\mathcal{A}, \mathtt{false}, c, m_0, m_2) \setminus \{ex_?\}$*:*

$$\left\{(\sigma \in \mathfrak{S})\ \dfrac{\vdash \Gamma}{\vdash \sigma(\Gamma)}\,ex\right\} \qquad \left\{\dfrac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}\,ex_t\right\} \qquad \left\{\dfrac{\vdash A, B, \Gamma}{\vdash B, A, \Gamma}\,ex_t^l\right\}$$

However note that (ex_t) and (ex_t^l) are not closed under composition (and (ex) is their closure under composition). We thus prefer (ex) (see Figure 2) as it allows us to have a more uniform treatment of exchanges together with closure under composition.

Let us mention that the cyclic exchange rule is strictly weaker than the general one since for example $A \otimes B \vdash B \otimes A$ (*i.e.* $\vdash B^\perp \parr A^\perp, B \otimes A$) is derivable if $p = \texttt{true}$ but not if $p = \texttt{false}$. In this spirit, the order of the context pieces in the $(\otimes)$ rule, which may look surprising to the reader used to commutative linear logic, happens to be the valid one for the cyclic case. If $p = \texttt{true}$, the $(\otimes)$ rule is equivalent to the more standard presentation:

$$\frac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta} \quad := \quad \frac{\dfrac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Delta, \Gamma}\otimes}{\vdash A \otimes B, \Gamma, \Delta}\, ex$$

Exponential Exchange Rule.

Together with the parameterized rule (ex), Figure 2 has an additional (exponential) exchange rule:

$$(\sigma \in \mathfrak{S})\ \frac{\vdash \Gamma, ?\Delta, \Sigma}{\vdash \Gamma, \sigma(?\Delta), \Sigma}\, ex_?$$

This $(ex_?)$ rule is clearly derivable from the (ex) rule if $p = \texttt{true}$. Thus it is redundant in this case. However, in the presence of the exponential connectives, some commutation of ?-formulas is required in the cyclic case ($p = \texttt{false}$) for the (cut) rule to be admissible since the rule:

$$\frac{\vdash ?A, ?B, \Gamma}{\vdash ?B, ?A, \Gamma}$$

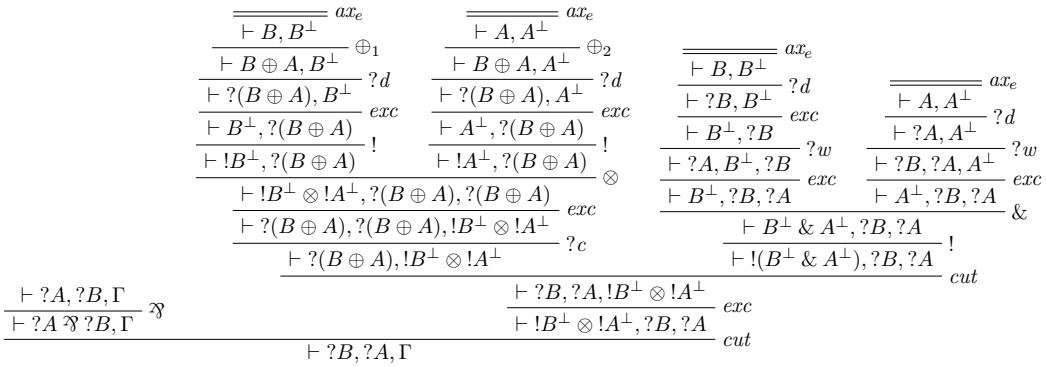
is derivable with the cut rule (see Figure 3, [addendum/ex_from_cut.v#comm_wn]). This comes from the fact that multiplicative connectives are related with additive connectives through the isomorphism $?A \parr ?B \simeq ?(A \oplus B)$ and the additive connectives are commutative independently of the exchange rules, so we get: $?A \parr ?B \simeq ?(A \oplus B) \simeq ?(B \oplus A) \simeq ?B \parr ?A$.

Alternative presentations of the $(ex_?)$ rule are possible and there are interesting relations between this rule and the structural rules available for the exponential connectives:

Lemma 5 (Exponential Exchange Rules [addendum/expexchange.v#ExponentialExchange])
The following (singleton) sets of rules are inter-derivable over $\texttt{LL}(\mathcal{A}, \texttt{false}, c, m_0, m_2) \setminus \{ex_?\}$*:*

$$\left\{(\sigma \in \mathfrak{S})\ \frac{\vdash \Gamma, ?\Delta, \Sigma}{\vdash \Gamma, \sigma(?\Delta), \Sigma}\, ex_?\right\} \qquad \left\{\frac{\vdash \Gamma, ?A, ?B, \Sigma}{\vdash \Gamma, ?B, ?A, \Sigma}\, ex_{?t}\right\}$$

$$\left\{\frac{\vdash ?\Gamma, ?\Gamma, \Delta}{\vdash ?\Gamma, \Delta}\, ?c_e\right\} \qquad \left\{\frac{\vdash ?A, ?\Gamma, ?A, \Delta}{\vdash ?A, ?\Gamma, ?A, \Delta}\, ?c^l\right\} \qquad \left\{\frac{\vdash ?A, ?\Gamma, ?A, \Delta}{\vdash ?\Gamma, ?A, \Delta}\, ?c^r\right\}$$



(we use here the (ax_e) rule, see Lemma 3)

Fig. 3 Commutation of ?-formulas in the cyclic systems.

The last three rules generalize the $?c$-rule in different (equivalent) ways. According to the remark above, these rules must be admissible to get cut admissibility (Figure 3). Our choice of $(ex_?)$ among them is driven by the quest of modularity and of minimising the impact over the usual presentation of rules in the commutative case, where all these rules are directly derivable. $(ex_?)$ is not exactly closed under composition, but consecutive applications which cannot be replaced by a single one always commute (which can be seen as a weaker form of modularity).

An alternative stronger variant can be considered where ?-formulas commute with all formulas (Yetter 1990), but it is not currently addressed in `Yalla` (see Appendix A for more comments). To conclude with non-commutative variants of linear logic, let us mention that systems discussed here rely on a unique linear negation connective $(\_)^\perp$. Considering two of them $(\_)^\perp$ and ${}^\perp(\_)$ leads to other non-commutative systems (Abrusci 1991) (not covered by `Yalla` so far).

The design choices for the rules of the $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ systems should now be clear, and we can start describing their properties.

2.2.2 Properties

The first list of properties we want to address are those leading to the cut admissibility property for $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$.

Parameter Sets.

If we order sets of axioms by inclusion and Booleans with `false` $<$ `true`, we can order tuples of parameters $(\mathcal{A}, p, c, m_0, m_2)$ component-wise. With respect to this *ordering*, provability in the induced systems is monotone. This remark allows us to factorize some proofs with respect to the values of the parameter set.

Proposition 1 (Monotonicity [ll_def.v#stronger_pfrag]) *If* $(\mathcal{A}, p, c, m_0, m_2) \leq (\mathcal{A}', p', c', m_0', m_2')$, *then* $\vdash \Gamma$ *provable in* $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ *implies* $\vdash \Gamma$ *provable in* $\mathsf{LL}(\mathcal{A}', p', c', m_0', m_2')$.

Constraints on the set of axioms $\mathcal{A}$ will be necessary for some properties to hold. We give here some names/notations for the most useful ones:

- $\mathcal{A}$ is *atomic* if all formulas in sequents of $\mathcal{A}$ are atomic;
- $\mathcal{A}$ is *cut-closed* if for any $\Gamma', A^\perp, \Gamma$ and Δ', A, Δ in $\mathcal{A}$, we have $\Gamma', \Delta, \Delta', \Gamma$ also in $\mathcal{A}$;
- if A is a formula, $A \notin \mathcal{A}$ means there is no list Γ, A, Γ' in $\mathcal{A}$.

In order to avoid repeating "$\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$" all the time, we will specify constraints on parameters required for statements to hold by using a list of constraints starting with the ⚠ symbol at the beginning of the statements requiring them (see for example Lemma 6 which holds only if $\perp$ does not occur in any list in $\mathcal{A}$).

Invertibility.

Invertibility results can be derived from cut admissibility (see Lemma 16). However it can also be interesting to give direct proofs which may require weaker hypotheses (so leading to stronger statements). Moreover some cut-admissibility proofs rely on invertibility properties to be proven first (Chaudhuri et al. 2017; Laurent 2018). For example:

Lemma 6 (Invertibility of $\perp$ [ll_def.v#bot_rev]) ⚠ $\perp \notin \mathcal{A}$
The rule $(\perp)$ *is invertible.*

Cut Admissibility.

We address the question of the admissibility of the cut rule in the general context of $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ with the different possible values of its parameters. The existence of the set $\mathcal{A}$ is an obstacle since, for example, if $\mathcal{A} = \{[X \,\&\, X]\}$ then $\vdash X$ is not cut-free provable (see [addendum/consistency.v#not_gax_cut_admissible]) but we have:

$$\dfrac{\dfrac{}{\vdash X \,\&\, X}\, ax_g \qquad \dfrac{\dfrac{}{\vdash \tilde{X}, X}\, ax}{\vdash \tilde{X} \oplus \tilde{X}, X}\, \oplus_1}{\vdash X}\, cut$$

We thus need to add restrictions on $\mathcal{A}$ for cut admissibility to hold. We will consider the case of atomic theories: sets of axioms containing atomic formulas only and closed under cut/transitivity, *i.e.* atomic and cut-closed sets of axioms.

Our proof follows a usual approach with exponential cuts eliminated through big substitution steps. This is similar to what happens in (Lincoln et al. 1992; Kanovich et al. 2019) for example. The proof uses an induction on a measure dealing with the size (Definition 1) of the cut formulas (*a.k.a.* the *degree* of the cuts) and the "size" of the proofs. There are two standard notions of sizes for sequent calculus proofs:

- the height of the proof, seen as a tree (count 1 for 0-ary rules, add 1 for unary rules, take the max and add 1 for binary rules);
- the number of rules (count 1 for 0-ary rules, add 1 for unary rules, take the sum and add 1 for binary rules).

They differ on binary rules (max *vs* sum). For our cut-admissibility proof, the exact choice of the notion of size of proofs is not very important. However in the more general context of the proof theory of linear logic, it happens that a mix of the two notions above is more appropriate: we take the sum of the sizes of the sub-proofs (+1) for the (*cut*), (⊗) and (*mix*)-rules (*i.e.* multiplicative binary rules), but the max (+1) for the (&)-rules (*i.e.* additive binary rules) [`ll_def.v#psize`]. All these three measures share the key property that a sub-proof π' of a proof π is strictly smaller than π. But the mixed version has also the property of being invariant under almost all rule permutations in linear logic. It mostly corresponds to computing the size of the biggest slice (Girard 1996) of the proof.

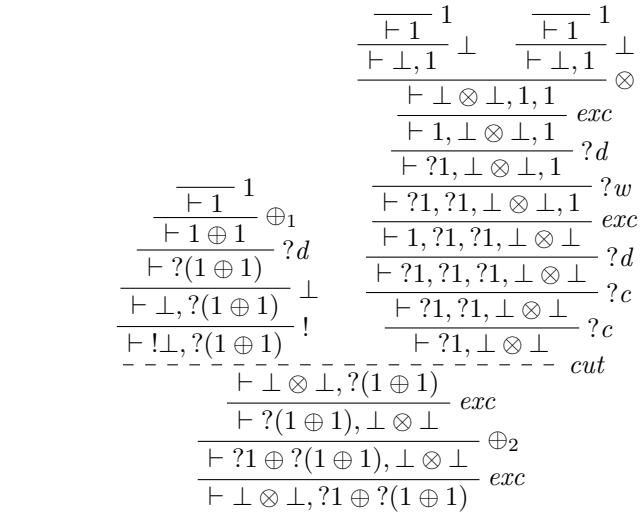
Before entering the technical details of the cut-admissibility proof, let us describe the key patterns on the following toy example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash 1 \oplus 1}\,\oplus_1}{\vdash ?(1 \oplus 1)}\,?d}{\vdash \bot, ?(1 \oplus 1)}\,\bot}{\vdash\, !\bot, ?(1 \oplus 1)}\,!}{\vdash ?(1 \oplus 1), !\bot}\,exc}{\vdash ?1 \oplus ?(1 \oplus 1), !\bot}\,\oplus_2}{\vdash\, !\bot, ?1 \oplus ?(1 \oplus 1)}\,exc \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash \bot, 1}\,\bot \qquad \dfrac{\dfrac{}{\vdash 1}\,1}{\vdash \bot, 1}\,\bot}{\vdash \bot \otimes \bot, 1, 1}\,\otimes}{\vdash 1, \bot \otimes \bot, 1}\,exc}{\vdash ?1, \bot \otimes \bot, 1}\,?d}{\vdash ?1, ?1, \bot \otimes \bot, 1}\,?w}{\vdash 1, ?1, ?1, \bot \otimes \bot}\,exc}{\vdash ?1, ?1, ?1, \bot \otimes \bot}\,?d}{\vdash ?1, ?1, \bot \otimes \bot}\,?c}{\vdash ?1, \bot \otimes \bot}\,?c}{\vdash \bot \otimes \bot, ?1 \oplus ?(1 \oplus 1)}\,cut$$

We start with a cut on the formulas $!\bot/?1$. Our goal is to turn the proof into a proof using cuts on pairs $\bot/1$ only. This is done in two steps:

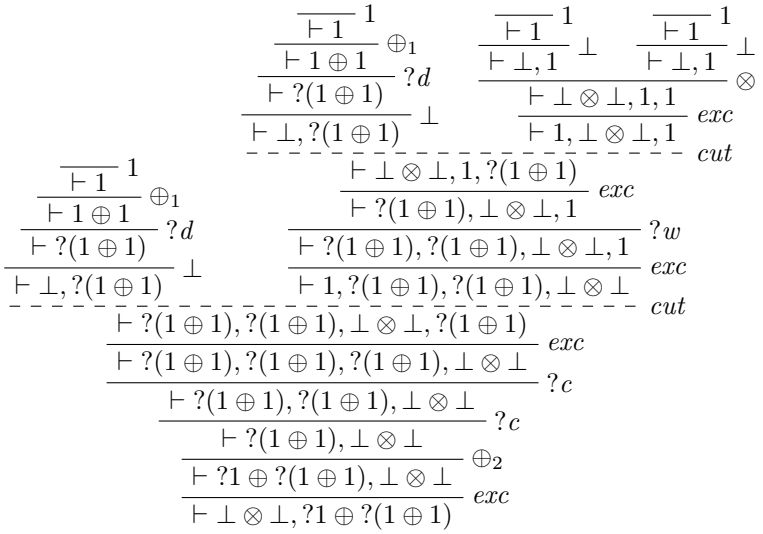
- *Commutation towards (!) rule:* in order to reduce a cut involving a ($?c$) rule, we need the other premise to end with a (!) rule[2]. This is done by commuting the cut up on the ! side until we reach a (!) rule.

[2]From a provability point of view, having the context of the other premise made of ?-formulas only is enough, but from a denotational semantics point of view it would be too weak: cut reduction would not preserve semantics.



The formal statement (Lemma 7) says something like: "if you know how to eliminate a cut on A with a ?-context, you can do it with a cut on $!A$ in any context".

- *Substitution of ?:* we look for the introductions of the ?-formulas involved in the $(?c)$ rules (here two $(?d)$ rules and one $(?w)$ rule), and we turn the cut into cuts on the premises of the $(?d)$ rules.



The formal statement (Lemma 8) is a bit more intricate since it must locate the various occurrences of the ?-formulas in a sequent in a way which is generic enough to deal with the non-commutative (cyclic) case (Kanovich et al. 2019).

We started with a single cut on the formulas $!\bot/?1$. We now reach a proof with two cuts on the strictly smaller formulas $\bot/1$. An induction hypothesis on the size of the cut formulas will then allow us to conclude and to get a cut-free proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash 1 \oplus 1}\,\oplus_1}{\vdash ?(1 \oplus 1)}\,?d \quad \dfrac{\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash 1 \oplus 1}\,\oplus_1}{\vdash ?(1 \oplus 1)}\,?d}{\vdash \bot, ?(1 \oplus 1)}\,\bot \quad \dfrac{}{\vdash \bot, ?(1 \oplus 1)}\,\bot}{\vdash \bot \otimes \bot, ?(1 \oplus 1), ?(1 \oplus 1)}\,\otimes}{\vdash ?(1 \oplus 1), \bot \otimes \bot, ?(1 \oplus 1)}\,exc}{\vdash ?(1 \oplus 1), ?(1 \oplus 1), \bot \otimes \bot, ?(1 \oplus 1)}\,?w}{\vdash ?(1 \oplus 1), ?(1 \oplus 1), ?(1 \oplus 1), \bot \otimes \bot}\,exc}{\vdash ?(1 \oplus 1), ?(1 \oplus 1), \bot \otimes \bot}\,?c}{\vdash ?(1 \oplus 1), \bot \otimes \bot}\,?c}{\vdash ?1 \oplus ?(1 \oplus 1), \bot \otimes \bot}\,\oplus_2}{\vdash \bot \otimes \bot, ?1 \oplus ?(1 \oplus 1)}\,exc$$

Here are now the formal key steps of the full cut-admissibility proof:

Lemma 7 (Commutation towards (!) rule [ll_cut.v#cut_oc_comm]) ⚠ atomic $\mathcal{A}$
Given some A, Γ_1 and Γ_2, if, for all Σ with $\vdash A, ?\Sigma$ we have $\vdash \Gamma_1, ?\Sigma, \Gamma_2$ then, for all Δ_1 and Δ_2, $\vdash \Delta_1, !A, \Delta_2$ entails $\vdash \Gamma_1, \Delta_2, \Delta_1, \Gamma_2$.

Proof By induction on the proof of $\vdash \Delta_1, !A, \Delta_2$. The only rule introducing a ! connective being the (!) rule, one can apply the assumption on Σ to the premise of the rule. □

Lemma 8 (Substitution of ? [ll_cut.v#substitution_oc]) ⚠ atomic $\mathcal{A}$
If the (cut) rule is admissible for the formula A, then from any proofs of $\vdash A^\perp, ?\Delta$ and $\vdash \Gamma_0, ?A, \Gamma_1, ?A, \ldots, \Gamma_{n-1}, ?A, \Gamma_n$, one can build a proof of $\vdash \Gamma_0, ?\Delta, \Gamma_1, ?\Delta, \ldots, \Gamma_{n-1}, ?\Delta, \Gamma_n$.

Proof By induction on the proof of $\vdash \Gamma_0, ?A, \Gamma_1, ?A, \ldots, \Gamma_{n-1}, ?A, \Gamma_n$, with the (*cut*) rule on A used in ($?d$) cases. Note the rule ($?c_e$) (Lemma 5) is used in the ($?c$) case:

$$\dfrac{\vdash ?A, ?A, \Gamma_1, ?A, \ldots, \Gamma_{n-1}, ?A, \Gamma_n}{\vdash ?A, \Gamma_1, ?A, \ldots, \Gamma_{n-1}, ?A, \Gamma_n}\,?c \quad \mapsto \quad \dfrac{\vdash ?A, ?A, \Gamma_1, ?A, \ldots, \Gamma_{n-1}, ?A, \Gamma_n}{\vdash ?\Delta, ?\Delta, \Gamma_1, ?\Delta, \ldots, \Gamma_{n-1}, ?\Delta, \Gamma_n}\,IH}{\vdash ?\Delta, \Gamma_1, ?\Delta, \ldots, \Gamma_{n-1}, ?\Delta, \Gamma_n}\,?c_e$$

□

In the commutative case, one can assume the copies of $?A$ to be consecutive, but in the cyclic case, one needs to consider this more general situation with an assumption $\vdash \Gamma_0, ?A, \Gamma_1, ?A, \ldots, \Gamma_{n-1}, ?A, \Gamma_n$ closed under cyclic exchange, for the induction to work.

Theorem 1 (Cut Admissibility [ll_cut.v#cut_r_gaxat]) ⚠ atomic and cut-closed $\mathcal{A}$
If $\vdash A^\perp, \Gamma$ and $\vdash A, \Delta$ are provable then $\vdash \Delta, \Gamma$ is provable.

Proof If $c = \mathtt{true}$, the result is immediate. Otherwise, we have to build a cut-free proof from two cut-free proofs.

We prove the slightly more general statement: if $\vdash A^\perp, \Gamma$ and $\vdash \Sigma, A, \Delta$ are provable then $\vdash \Sigma, \Gamma, \Delta$ is provable. We rely on a trick which swaps the two premises sometimes in the proof to avoid generalizing the context on the $A^\perp$ side as well.

We go by induction on the lexicographically ordered pair (c, s) where c is the size of the cut formula A and s is the sum of the sizes of the proofs of $\vdash A^\perp, \Gamma$ and $\vdash \Sigma, A, \Delta$.

- If A is not active in the last rule of $\vdash \Sigma, A, \Delta$, we commute the cut up on this side and get smaller premises to apply the induction hypothesis on s.
- If A is active in the last rule of $\vdash \Sigma, A, \Delta$ (thus Σ is empty or we have an axiom rule), and $A^\perp$ is not active in $\vdash A^\perp, \Gamma$, then $A^\perp$ must be involved in an exchange (or (mix)) rule. If Σ is empty, we commute the cut up with this rule and we apply the induction hypothesis on s with swapped premises (in the case of an $(ex_?)$ rule, we use Lemma 7). If we have an axiom rule introducing A, we use an auxiliary induction on the proof of $\vdash A^\perp, \Gamma$.
- If both A and $A^\perp$ are active in their rules, we consider the formula A itself:
  - If it is an atomic formula, the premises must be (ax) or (ax_g) rules. If one of them is an (ax) rule, we immediately get the other premise as a cut-free proof of the corresponding sequent. If both are (ax_g) rules, we replace them by a single application of the (ax_g) rule thanks to cut-closedness of $\mathcal{A}$.
  - If it is an exponential formula, we apply Lemma 8.
  - Otherwise we are in a multiplicative or additive key case in which we can replace the cut by cuts on strict sub-formulas of A. The $\otimes/\parr$ case generates two cuts for example:

$$\dfrac{\dfrac{\vdash B^\perp, \Delta \qquad \vdash A^\perp, \Gamma}{\vdash B^\perp \otimes A^\perp, \Gamma, \Delta}\otimes \qquad \dfrac{\vdash A, B, \Sigma}{\vdash A \parr B, \Sigma}\parr}{\vdash \Sigma, \Gamma, \Delta}\,cut \quad \mapsto \quad \dfrac{\vdash B^\perp, \Delta \qquad \dfrac{\vdash A^\perp, \Gamma \qquad \vdash A, B, \Sigma}{\vdash B, \Sigma, \Gamma}\,cut}{\vdash \Sigma, \Gamma, \Delta}\,cut$$

  where the order of formulas and the fact that $(A \parr B)^\perp = B^\perp \otimes A^\perp$ (in this specific order) is important. □

We have shown cut admissibility for $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$ (under appropriate assumptions on $\mathcal{A}$, which hold in particular if $\mathcal{A} = \emptyset$). It is often compared with the more "constructive" *cut-elimination* property, considered as an algorithm which builds a cut-free proof by applying small reduction steps in a proof with cuts. Note that our proof here is constructive and thus provides an algorithm to compute a cut-free proof. A closer look at the proof allows to identify the small transformation steps involved. However it would be slightly different, but useful for a fine grained analysis of proof transformations, to provide explicitly a rewriting system on proofs for small cut-reduction steps and to prove a termination property. We do not address this here.

Conservativity over fragments defined by restricting the set of formulas is a key property of cut-free provability in a propositional setting. Given a set $\mathcal{F}$ of formulas, one can define an associated sub-system (of linear logic) by restricting the proofs so that they use formulas from $\mathcal{F}$ only.

Proposition 2 (Conservativity [ll_prop.v#conservativity]) ⚠ $c = \texttt{false}$
If $\mathcal{F}$ is a set of formulas closed under sub-formula, and Γ contains formulas from $\mathcal{F}$ only, then $\vdash \Gamma$ is provable in the full system if and only if it is provable in the fragment associated with $\mathcal{F}$.

Proof By induction on cut-free proofs. The premises of each (non-cut) rule contain only sub-formulas of the formulas in its conclusion. If all the formulas of the conclusion of a rule are in $\mathcal{F}$, then all the formulas in the premises as well, and we can apply the induction hypothesis. □

An alternative way of stating this property is the sub-formula property.

Proposition 3 (Sub-Formula Property [ll_prop.v#subformula]) ⚠ $c = \texttt{false}$
A proof of $\vdash \Gamma$ is provable then it has a proof containing sub-formulas of formulas of Γ only.

Proof The set of all sub-formulas of Γ is closed under sub-formula. By Proposition 2, there is a proof of $\vdash \Gamma$ with sub-formulas of Γ only. □

Deduction.

The deduction lemma relates the possibility of deriving a sequent $\vdash \Gamma$ in a finite theory $\mathcal{T}$ (*i.e.* of building an open proof from admitted sequents in $\mathcal{T}$) with the provability of Γ enriched with formulas in $\mathcal{T}$. In linear logic, this requires to introduce ?-connectives (see for example (Avron 1988, Modal Deduction Theorem)). As already mentioned in Section 2.2, we use $\mathcal{A}$ to implement the use of theories.

Lemma 9 (?-Extension [ll_def.v#ext_wn]) ⚠ $p = \texttt{true}$
If $\vdash \Gamma$ is provable in $\mathsf{LL}(\mathcal{A}, \texttt{true}, c, m_0, m_2)$ then, by defining $\mathcal{A}' = \{\Delta \multimap ?\Sigma \mid \Delta \in \mathcal{A}\}$, we have $\vdash \Gamma, ?\Sigma$ provable in $\mathsf{LL}(\mathcal{A}', \texttt{true}, c, m_0, m_2)$.

Proof By induction on the proof of $\vdash \Gamma$:

- If we have a (ax_g) rule the result is immediate by assumption.
- For other 0-ary rules, we use $(?w)$ rules.
- For (!) we can apply the induction hypothesis, since the context stays with ?-formulas only.
- For other 1-ary rules, we directly apply the induction hypothesis.
- For 2-ary rules, we use $(?c)$ rules (except for the (&) rule). □

Theorem 2 (Deduction [ll_prop.v#deduction*]) ⚠ $p = \texttt{true}$ and $c = \texttt{true}$ $\vdash \Gamma, ?A_1^\perp, \ldots, ?A_n^\perp$ *is provable if and only if there exists an open proof of* $\vdash \Gamma$ *where the open hypotheses are either from* $\mathcal{A}$ *or of the shape* $\vdash A_i$ $(1 \leq i \leq n)$ *(*i.e. $\vdash \Gamma$ *is provable in* $\mathsf{LL}(\mathcal{A} \cup \{[A_1], \ldots, [A_n]\}, \texttt{true}, \texttt{true}, m_0, m_2)$*).*

Proof Assuming that $\vdash \Gamma, ?A_1^\perp, \ldots, ?A_n^\perp$ is provable, we can introduce cuts with (ax_g) rules. Here is the case $n = 1$, starting from a proof of $\vdash \Gamma, ?A^\perp$, we can build:

$$\dfrac{\dfrac{\vdash \Gamma, ?A^\perp}{\vdash ?A^\perp, \Gamma}\,exc \qquad \dfrac{\dfrac{}{\vdash A}\,ax_g}{\vdash !A}\,!}{\vdash \Gamma}\,cut$$

Conversely, we can apply Lemma 9 to build a proof of $\vdash \Gamma, ?A_1^\perp, \ldots, ?A_n^\perp$ in $\mathsf{LL}(\mathcal{A}', \texttt{true}, \texttt{true}, m_0, m_2)$ where $\mathcal{A}' = \{\Delta, ?A_1^\perp, \ldots, ?A_n^\perp \mid \Delta \in \mathcal{A}\} \cup \{A_i, ?A_1^\perp, \ldots, ?A_n^\perp \mid 1 \leq i \leq n\}$. We modify each occurrence of (ax_g) in the obtained proof to get the proof of $\vdash \Gamma, ?A_1^\perp, \ldots, ?A_n^\perp$ in $\mathsf{LL}(\mathcal{A}, \texttt{true}, \texttt{true}, m_0, m_2)$. This depends on which part of the union is used from $\mathcal{A}'$:

$$(\Delta \in \mathcal{A})\ \dfrac{\dfrac{\dfrac{}{\vdash \Delta}\,ax_g}{\vdash ?A_1^\perp, \ldots, ?A_n^\perp, \Delta}\,?w}{\vdash \Delta, ?A_1^\perp, \ldots, ?A_n^\perp}\,exc$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A_i, A_i^\perp}\,ax_e}{\vdash ?A_{i+1}^\perp, \ldots, ?A_n^\perp, A_i, A_i^\perp}\,?w}{\vdash A_i^\perp, ?A_{i+1}^\perp, \ldots, ?A_n^\perp, A_i}\,exc}{\vdash ?A_i^\perp, ?A_{i+1}^\perp, \ldots, ?A_n^\perp, A_i}\,?d}{\vdash ?A_1^\perp, \ldots, ?A_n^\perp, A_i}\,?w}{\vdash A_i, ?A_1^\perp, \ldots, ?A_n^\perp}\,exc$$

□

The result is restricted here to the commutative case ($p = \texttt{true}$). It should hold in the cyclic case with the strong exponential exchange rule (see Appendix A).

2.2.3 Mix Rules

When writing LL, we mean $\mathsf{LL}(\emptyset, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{false})$, *i.e.* core commutative linear logic, for which (*cut*) is admissible (Theorem 1) but it is not a rule of the system. The (mix_0) and (*mix*) rules (Girard 1987) are usually not part of core linear logic but nevertheless they occur quite often in the literature (for example (Fleury and Retoré 1994; Bellin 1997; Pagani 2007; Nguyen 2020; Laurent 2026)).

Let us first comment on the (possibly surprising) (mix_0) rule. In a logical system which allows weakening on arbitrary formulas $\dfrac{\vdash \Gamma}{\vdash A, \Gamma}$, the provability of the empty sequent $\vdash$ entails the provability of all sequents (the converse being immediate). However in linear logic, this is *not* the case. This leads to two different notions of inconsistency: weak and strong inconsistencies. They can be characterised in various ways, as shown by Lemmas 10 and 11.

Lemma 10 (Weak Inconsistency [addendum/consistency.v#*empty*]) *Using the (cut) rule and the multiplicative* LL *rules, the following statements are equivalent:*

(i) $\vdash$ *is provable;*
(ii) $\vdash \bot, \bot$ *is provable (*i.e. $1 \vdash \bot$ *is provable);*
(iii) there exists a formula A *such that both* $\vdash A$ *and* $\vdash A^\bot$ *are provable.*

Lemma 11 (Strong Inconsistency [addendum/consistency.v#*zero*]) *Using the (cut) rule, the following statements are equivalent:*

(i) $\vdash 0$ *is provable;*
(ii) for any Γ, $\vdash \Gamma$ *is provable.*

Strong inconsistency entails weak inconsistency (if all sequents are provable, then in particular $\vdash$ is provable), but the converse is wrong. Core linear logic is strongly consistent (*i.e.* not weakly inconsistent) and its extension with (mix_0) is weakly consistent (*i.e.* weakly inconsistent but not strongly inconsistent).

Let us now analyse the expressive power of the mix rules. It is possible to relate the provability in extensions of LL with mix rules with the provability in the core system.

Lemma 12 (mix_0 [ll_fragments.v#*mix0*]) ⚠ $p =$ `true`
Using the (cut) rule, the following statements are equivalent:

(i) $\vdash \Gamma$ *is provable in* LL *with* (mix_0);
(ii) $\vdash \Gamma$ *is provable in* LL *extended with the axiom* $\vdash \bot, \bot$ *(*i.e. *the provability of* $1 \vdash \bot$*).*

Lemma 13 (mix [ll_fragments.v#*mix2*]) ⚠ $p =$ `true`
Using the (cut) rule, the following statements are equivalent:

(i) $\vdash \Gamma$ *is provable in* LL *with (mix);*
(ii) $\vdash \Gamma$ *is provable in* LL *extended with the axiom* $\vdash 1, 1$ *(*i.e. *the provability of* $\bot \vdash 1$*);*
(iii) $\vdash \Gamma$ *is provable in* LL *extended with the axioms* $\vdash B^\bot \parr A^\bot, A \parr B$ *for all* A *and* B *(*i.e. *the provability of* $A \otimes B \vdash A \parr B$*).*

See Appendix B for more comments on (iii).

3 Intuitionistic Linear Logic

The study of intuitionistic linear logic (Girard and Lafont 1987) follows a very similar pattern to the one followed for classical linear logic. We thus go faster on the basic points.

3.1 Formulas and Sequents

Given a set `IAtom` (whose elements are denoted X, Y, etc.) and a fixed element `N` (used for negations) of `IAtom`, *formulas* of propositional intuitionistic linear logic are

defined by the grammar:

$$A ::= X \mid 1 \mid A \otimes A \mid A \multimap A \mid A \circ\!\!- A \mid \neg A \mid A\bot \mid 0 \mid \top \mid A \oplus A \mid A \,\&\, A \mid !A$$

The use of two different implication (resp. negation) connectives is related with non-commutativity (see Section 3.2.1).

`N` is used to avoid sequents with an empty right-hand side. Sequents are given as a pair of a list of formulas Γ and of a formula A, *i.e.* $\Gamma \vdash A$. Sequents of the shape $\Gamma \vdash$ `N` can then be interpreted as sequents with an empty right-hand side, *i.e.* $\Gamma \vdash$. Indeed only variants of the rules ($\neg$L), ($\bot$L) and (0L) would introduce sequents $\Gamma \vdash$ with an empty right-hand side, and this is replaced with $\Gamma \vdash$ `N` (Figure 4). As a consequence, `N` has to be considered as a sub-formula of $\neg A$ and $A\bot$ for the sub-formula property to hold.

3.2 Proofs

The sequent calculus system $\mathsf{ILL}(\mathcal{I}, p, c)$ for intuitionistic linear logic depends on *three parameters*:

- $\mathcal{I}$ is a set of pairs of lists of formulas and formulas (considered as valid sequents in the system, *i.e.* axioms);
- p is a Boolean controlling the impact of the exchange rule (thus commutativity);
- c is a Boolean controlling whether or not a primitive *cut* rule belongs to the system.

The *rules of* $\mathsf{ILL}(\mathcal{I}, p, c)$ are given in Figure 4 with:

$$\mathfrak{I}_p = \begin{cases} \mathfrak{S} & \text{if } p = \texttt{true} \\ \{id\} & \text{if } p = \texttt{false} \end{cases}$$

where id is the identity permutation. Alternative equivalent choices are possible, like $\mathfrak{I}_{\texttt{false}} = \emptyset$, or using p to simply control whether the rule can be applied or not (as done with c for the (*cut*) rule). But we prefer the present approach to get something uniform with what happens in the classical case, and to have smoother (parameterized) relations between $\mathfrak{S}_p$ and $\mathfrak{I}_p$ (see Theorem 4).

Lemma 14 (Axiom Expansion [`ill_def.v#ax_exp_ill`]) *Let A be a formula, the extended axiom rule* $\dfrac{}{A \vdash A}\, ax_e$ *is derivable.*

No exchange rule at all is necessary for this lemma.

Parameter Sets.

If we order sets of axioms by inclusion and Booleans with `false` $<$ `true`, we can order triples of parameters $(\mathcal{I}, p, c)$ component-wise. With respect to this *ordering*, provability in the induced systems is monotone:

$$\frac{}{X \vdash X}\ ax \qquad ((\Gamma, C) \in \mathcal{I})\ \frac{}{\Gamma \vdash C}\ ax_g \qquad (c)\ \frac{\Delta \vdash A \qquad \Gamma, A, \Sigma \vdash C}{\Gamma, \Delta, \Sigma \vdash C}\ cut$$

$$(\sigma \in \mathfrak{I}_p)\ \frac{\Gamma \vdash C}{\sigma(\Gamma) \vdash C}\ ex \qquad (\sigma \in \mathfrak{S})\ \frac{\Gamma, !\Delta, \Sigma \vdash C}{\Gamma, \sigma(!\Delta), \Sigma \vdash C}\ ex_!$$

$$\frac{}{\vdash 1}\ 1\mathrm{R} \qquad \frac{\Gamma, \Delta \vdash C}{\Gamma, 1, \Delta \vdash C}\ 1\mathrm{L}$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ \otimes\mathrm{R} \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \otimes B, \Delta \vdash C}\ \otimes\mathrm{L}$$

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\ \multimap \mathrm{R} \qquad \frac{\Delta \vdash A \qquad \Gamma, B, \Sigma \vdash C}{\Gamma, \Delta, A \multimap B, \Sigma \vdash C}\ \multimap \mathrm{L}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash B \circ\!\!- A}\ \circ\!\!- \mathrm{R} \qquad \frac{\Delta \vdash A \qquad \Gamma, B, \Sigma \vdash C}{\Gamma, B \circ\!\!- A, \Delta, \Sigma \vdash C}\ \circ\!\!- \mathrm{L}$$

$$\frac{A, \Gamma \vdash \mathtt{N}}{\Gamma \vdash \neg A}\ \neg\mathrm{R} \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \mathtt{N}}\ \neg\mathrm{L} \qquad \frac{\Gamma, A \vdash \mathtt{N}}{\Gamma \vdash A \urcorner}\ \urcorner\mathrm{R} \qquad \frac{\Gamma \vdash A}{A \urcorner, \Gamma \vdash \mathtt{N}}\ \urcorner\mathrm{L}$$

$$\frac{}{\Gamma \vdash \top}\ \top \qquad \frac{}{\Gamma, 0, \Delta \vdash C}\ 0\mathrm{L}$$

$$\frac{\Gamma, A, \Delta \vdash C}{\Gamma, A \,\&\, B, \Delta \vdash C}\ \&_1\mathrm{L} \qquad \frac{\Gamma, A, \Delta \vdash C}{\Gamma, B \,\&\, A, \Delta \vdash C}\ \&_2\mathrm{L} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \&\mathrm{R}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ \oplus_1\mathrm{R} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash B \oplus A}\ \oplus_2\mathrm{R} \qquad \frac{\Gamma, A, \Delta \vdash C \qquad \Gamma, B, \Delta \vdash C}{\Gamma, A \oplus B, \Delta \vdash C}\ \oplus\mathrm{L}$$

$$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\ !\mathrm{R} \qquad \frac{\Gamma, A, \Delta \vdash C}{\Gamma, !A, \Delta \vdash C}\ !d\mathrm{L} \qquad \frac{\Gamma, \Delta \vdash C}{\Gamma, !A, \Delta \vdash C}\ !w\mathrm{L}$$

$$\frac{\Gamma, !A, !A, \Delta \vdash C}{\Gamma, !A, \Delta \vdash C}\ !c\mathrm{L}$$

Fig. 4 The $\mathsf{ILL}(\mathcal{I}, p, c)$ system.

Proposition 4 (Monotonicity [ill_def.v#stronger_ipfrag]) *If* $(\mathcal{I}, p, c) \leq (\mathcal{I}', p', c')$*, then* $\Gamma \vdash A$ *provable in* $\mathsf{ILL}(\mathcal{I}, p, c)$ *implies* $\Gamma \vdash A$ *provable in* $\mathsf{ILL}(\mathcal{I}', p', c')$.

Constraints on the set of axioms $\mathcal{I}$ will be necessary for some properties to hold. We give here some useful ones:

- $\mathcal{I}$ is *N-free* (denoted $\mathtt{N} \notin_l \mathcal{I}$) if $\mathtt{N}$ does not occur as a formula in the left-hand side of sequents of $\mathcal{I}$ (it may however be a strict sub-formula or occur on the right-hand side);
- $\mathcal{I}$ is *atomic* if all formulas in sequents of $\mathcal{I}$ are atomic (*i.e.* belong to $\mathtt{IAtom}$);
- $\mathcal{I}$ is *cut-closed* if for any (Δ, A) and $(\Gamma \mathbin{⅋} [A] \mathbin{⅋} \Sigma, B)$ in $\mathcal{I}$, $(\Gamma \mathbin{⅋} \Delta \mathbin{⅋} \Sigma, B)$ is also in $\mathcal{I}$.

In order to avoid repeating "$\mathsf{ILL}(\mathcal{I}, p, c)$" all the time, we will again use the ⚠ symbol to describe constraints on the parameters.

3.2.1 Exchange Rules and Non-Commutative Logic

We cannot restrict left rules to act on formulas in head-position only in the left-hand side of sequents since, in the non-commutative case ($p = \mathtt{false}$), there is no way to permute formulas (as opposed to what happened in the classical case where cyclic exchange is always available in the considered variants).

In the intuitionistic setting, cyclic exchange generates full exchange through cut admissibility:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e \quad \dfrac{}{B \vdash B}\,ax_e}{A, B \vdash A \otimes B}\,\otimes\mathrm{R}}{B, A \vdash A \otimes B}\,exc \qquad \dfrac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \otimes B, \Delta \vdash C}\,\otimes\mathrm{L}}{\Gamma, B, A, \Delta \vdash C}\,cut$$

The right-hand side formula acts as a fixed point during cyclic exchange in intuitionistic systems, so that moving from $A, B \vdash C$ to $B, A \vdash C$ is like moving from (A, B, C) to (B, A, C) which contains the full power of transpositions.

By restricting $\mathsf{ILL}(\emptyset, \mathtt{false}, \mathtt{false})$ to the connectives $\otimes$, $\multimap$ and $\circ\!\!-$, one exactly gets Lambek calculus (Lambek 1958). This is one of the reasons for considering non-commutativity in ILL. Thanks to cut admissibility and the induced sub-formula property, $\mathsf{ILL}(\emptyset, \mathtt{false}, \mathtt{true})$ is a conservative extension of Lambek calculus (Proposition 5). In particular, as in Lambek calculus, the two implications are different. They become equivalent only in the commutative case, thanks to exchange:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e \quad \dfrac{}{B \vdash B}\,ax_e}{A, A \multimap B \vdash B}\,\multimap\mathrm{L}}{A \multimap B, A \vdash B}\,ex}{A \multimap B \vdash B \circ\!\!- A}\,\circ\!\!-\mathrm{R} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e \quad \dfrac{}{B \vdash B}\,ax_e}{B \circ\!\!- A, A \vdash B}\,\multimap\mathrm{L}}{A, B \circ\!\!- A \vdash B}\,ex}{B \circ\!\!- A \vdash A \multimap B}\,\circ\!\!-\mathrm{R}$$

Concerning the exponential connectives, similarly to what happens with classical linear logic (Figure 3), the commutation of ! formulas is required, even in the non-commutative case ($p = \texttt{false}$), for the (cut) rule to be admissible:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e}{!A \vdash A}\,!d\text{L}}{!B, !A \vdash A}\,!w\text{L} \quad \dfrac{\dfrac{\dfrac{}{B \vdash B}\,ax_e}{!B \vdash B}\,!d\text{L}}{!B, !A \vdash B}\,!w\text{L}}{!B, !A \vdash A \,\&\, B}\,\&\text{R}}{!B, !A \vdash !(A \,\&\, B)}\,!\text{R} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e}{A \,\&\, B \vdash A}\,\&_1\text{L}}{!(A \,\&\, B) \vdash A}\,!d\text{L}}{!(A \,\&\, B) \vdash !A}\,!\text{R} \quad \dfrac{\dfrac{\dfrac{}{B \vdash B}\,ax_e}{A \,\&\, B \vdash B}\,\&_2\text{L}}{!(A \,\&\, B) \vdash B}\,!d\text{L}}{!(A \,\&\, B) \vdash !B}\,!\text{R}}{!(A \,\&\, B), !(A \,\&\, B) \vdash !A \otimes !B}\,\otimes\text{R}}{!(A \,\&\, B) \vdash !A \otimes !B}\,!c\text{L}}{!B, !A \vdash !A \otimes !B}\,cut \qquad \dfrac{\Gamma, !A, !B, \Delta \vdash C}{\Gamma, !A \otimes !B, \Delta \vdash C}\,\otimes\text{L}}{\Gamma, !B, !A, \Delta \vdash C}\,cut$$

This is why the ($ex_!$) rule is included in the system.

3.2.2 Implication and Negation

There are strong connections between the negation connectives and the implication connectives. We state here the properties for $\neg$ and $\multimap$ but everything mentioned in the current section applies similarly to $\rightarrowtail$ and $\circ\!\!-$. One can prove $\neg A \vdash A \multimap \mathtt{N}$ and $A \multimap \mathtt{N} \vdash \neg A$ (and similarly $A \rightarrowtail \vdash \mathtt{N} \circ\!\!- A$ and $\mathtt{N} \circ\!\!- A \vdash A \rightarrowtail$):

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e}{A, \neg A \vdash \mathtt{N}}\,\neg\text{L}}{\neg A \vdash A \multimap \mathtt{N}}\,\multimap\text{R} \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax_e \quad \dfrac{}{\mathtt{N} \vdash \mathtt{N}}\,ax}{A, A \multimap \mathtt{N} \vdash \mathtt{N}}\,\multimap\text{L}}{A \multimap \mathtt{N} \vdash \neg A}\,\neg\text{R}$$

This means that, *using cuts*, it is always possible to replace one of these constructions by the other. Assuming we do not want to introduce cuts, it seems that $A \multimap \mathtt{N}$ may induce more proofs than $\neg A$ because of the richer left rule (additional possible contexts):

$$\dfrac{\Delta \vdash A \qquad \Gamma, \mathtt{N}, \Sigma \vdash C}{\Gamma, \Delta, A \multimap \mathtt{N}, \Sigma \vdash C}\,\multimap\text{L}$$

but one can prove, *without introducing cuts*, that in this case we also have a proof of $\Gamma, \Delta, \neg A, \Sigma \vdash C$:

Lemma 15 (Extended left rule for $\neg$ [ill_def.v#neg_map_rule]) ⚠ $\mathtt{N} \notin_l \mathcal{I}$
The following rule is admissible:

$$\dfrac{\Delta \vdash A \qquad \Gamma, \mathtt{N}, \Sigma \vdash C}{\Gamma, \Delta, \neg A, \Sigma \vdash C}$$

All this suggests that, instead of considering a primitive connective $\neg$, we could define $\neg A$ as $A \multimap \mathtt{N}$, and derive the rules for $\neg$ from the rules for $\multimap$:

$$\frac{A, \Gamma \vdash \mathtt{N}}{\Gamma \vdash \neg A} \multimap \mathrm{R} \qquad\qquad \frac{\Gamma \vdash A \qquad \dfrac{}{\mathtt{N} \vdash \mathtt{N}}\, ax}{\Gamma, \neg A \vdash \mathtt{N}} \multimap \mathrm{L}$$

Since some variants of intuitionistic linear logic like tensor logic TL (Section 5.1) use a primitive negation connective, we prefer introducing negation connectives in ILL directly. In this way TL is a fragment of ILL, and the relation between $\neg$ and $\multimap$ can be studied inside ILL.

3.2.3 Properties

The proof of the admissibility of the (*cut*) rule follows a similar pattern as for the classical case.

Theorem 3 (Cut Admissibility [ill_cut.v#cut_ir_gaxat]) ⚠ $\mathtt{N} \notin_l \mathcal{I}$, atomic and cut-closed $\mathcal{I}$
If $\Delta \vdash A$ and $\Gamma, A, \Sigma \vdash B$ are provable then $\Gamma, \Delta, \Sigma \vdash B$ is provable.

Remark. The hypothesis $\mathtt{N} \notin_l \mathcal{I}$ in Theorem 3 is required to avoid the situation:

$$\frac{\dfrac{\Delta \vdash A}{\Delta, \neg A \vdash \mathtt{N}}\, \neg\mathrm{L} \qquad \dfrac{}{\Gamma, \mathtt{N}, \Sigma \vdash B}\, ax_g}{\Gamma, \Delta, \neg A, \Sigma \vdash B}\, cut$$

where we are stuck.

Lemma 16 (Invertibility [ill_prop.v#*_rev*_noax]) ⚠ $\mathcal{I} = \emptyset$
The rules ($1\mathrm{L}$), ($\otimes\mathrm{L}$), ($\multimap\mathrm{R}$), ($\circ\!\!-\mathrm{R}$), ($\neg\mathrm{R}$), ($\parr\mathrm{R}$), ($\&\mathrm{R}$), ($\oplus\mathrm{L}$), ($!\mathrm{R}$) are invertible.

Proof For example:

$$\frac{\Gamma \vdash A \,\&\, B \qquad \dfrac{\dfrac{}{A \vdash A}\, ax_e}{A \,\&\, B \vdash A}\, \&_1\mathrm{L}}{\Gamma \vdash A}\, cut \qquad \text{and} \qquad \frac{\Gamma \vdash A \,\&\, B \qquad \dfrac{\dfrac{}{B \vdash B}\, ax_e}{A \,\&\, B \vdash B}\, \&_2\mathrm{L}}{\Gamma \vdash B}\, cut$$

□

Proposition 5 (Conservativity [ill_prop.v#iconservativity_axfree]) ⚠ $c =$ `false`
If $\mathcal{F}$ is a set of formulas closed under sub-formula, and Γ and A contain formulas from $\mathcal{F}$ only, then $\Gamma \vdash A$ is provable in the full system if and only if it is provable in the fragment associated with $\mathcal{F}$.

Proposition 6 (Sub-Formula Property [ill_prop.v#isubformula]) ⚠ $c =$ `false`
If $\Gamma \vdash A$ is provable then it has a proof containing sub-formulas of formulas of Γ and A only.

4 Between LL and ILL

There is a standard direct embedding of (commutative) ILL into LL which consists in translating $A \multimap B$ into $A^\perp \parr B$, and $\Gamma \vdash C$ into $\vdash \Gamma^\perp, C$. To be consistent with the non-commutative settings, one needs to be a bit more careful and to consider $\Gamma \vdash C \mapsto \ \vdash C, \overline{\Gamma}^\perp$. One has to be careful with the translation of multiplicative formulas as well.

Definition 2 (Embedding of Formulas [ill_vs_ll.v#ill2ll]) Given a function ι from `IAtom` to `Atom`, we extend it into a function from intuitionistic formulas to classical formulas:

$$\begin{array}{rcl@{\qquad}rcl}
\iota(1) &:=& 1 & & & \\
\iota(A \otimes B) &:=& \iota(A) \otimes \iota(B) & & & \\
\iota(A \multimap B) &:=& \iota(A)^\perp \parr \iota(B) & \iota(B \circ\!\!- A) &:=& \iota(B) \parr \iota(A)^\perp \\
\iota(\neg A) &:=& \iota(A)^\perp \parr \iota(\mathtt{N}) & \iota(A \multimap) &:=& \iota(\mathtt{N}) \parr \iota(A)^\perp \\
\iota(0) &:=& 0 & \iota(\top) &:=& \top \\
\iota(A \oplus B) &:=& \iota(A) \oplus \iota(B) & \iota(A \,\&\, B) &:=& \iota(A) \,\&\, \iota(B) \\
\iota(!A) &:=& !\iota(A) & & &
\end{array}$$

We use the notation $\iota(\mathcal{I})$ for $\{\iota(D), \overline{\iota(\Delta)}^\perp \mid (\Delta, D) \in \mathcal{I}\}$.

Theorem 4 (Embedding [ill_vs_ll.v#ill_to_ll]) *If $\Gamma \vdash C$ is provable in $\mathsf{ILL}(\mathcal{I}, p, c)$ then $\vdash \iota(C), \overline{\iota(\Gamma)}^\perp$ is provable in $\mathsf{LL}(\iota(\mathcal{I}), p, c, \mathtt{false}, \mathtt{false})$.*

Proof We give a few key cases:

$$((\Gamma, C) \in \mathcal{I})\ \frac{}{\Gamma \vdash C}\ ax_g \qquad \mapsto \qquad (\iota(C), \overline{\iota(\Gamma)}^\perp \in \iota(\mathcal{I}))\ \frac{}{\vdash \iota(C), \overline{\iota(\Gamma)}^\perp}\ ax_g$$

$$(\sigma \in \mathfrak{I}_p)\ \frac{\Gamma \vdash C}{\sigma(\Gamma) \vdash C}\ ex \qquad \mapsto \qquad (\sigma \in \mathfrak{S}_p)\ \frac{\vdash \iota(C), \overline{\iota(\Gamma)}^\perp}{\vdash \iota(C), \overline{\sigma(\iota(\Gamma))}^\perp}\ ex$$

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\ \multimap \mathrm{R} \qquad \mapsto \qquad \frac{\dfrac{\vdash \iota(B), \overline{\iota(\Gamma)}^\perp, \iota(A)^\perp}{\vdash \iota(A)^\perp, \iota(B), \overline{\iota(\Gamma)}^\perp}\ exc}{\vdash \iota(A)^\perp \parr \iota(B), \overline{\iota(\Gamma)}^\perp}\ \parr$$

$$\frac{\Delta \vdash A \qquad \Gamma, B, \Sigma \vdash C}{\Gamma, B \circ\!\!- A, \Delta, \Sigma \vdash C}\ \circ\!\!- \mathrm{L}$$

$$\mapsto \qquad \frac{\dfrac{\vdash \iota(A), \overline{\iota(\Delta)}^\perp \qquad \dfrac{\vdash \iota(C), \overline{\iota(\Sigma)}^\perp, \iota(B)^\perp, \overline{\iota(\Gamma)}^\perp}{\vdash \iota(B)^\perp, \overline{\iota(\Gamma)}^\perp, \iota(C), \overline{\iota(\Sigma)}^\perp}\ exc}{\vdash \iota(A) \otimes \iota(B)^\perp, \overline{\iota(\Gamma)}^\perp, \iota(C), \overline{\iota(\Sigma)}^\perp, \overline{\iota(\Delta)}^\perp}\ \otimes}{\vdash \iota(C), \overline{\iota(\Sigma)}^\perp, \overline{\iota(\Delta)}^\perp, \iota(A) \otimes \iota(B)^\perp, \overline{\iota(\Gamma)}^\perp}\ exc$$

$$\frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \mathtt{N}}\ \neg\mathrm{L} \qquad \mapsto \qquad \dfrac{\dfrac{\dfrac{}{\vdash \iota(\mathtt{N})^\perp, \iota(\mathtt{N})}\ ax \qquad \vdash \iota(A), \overline{\iota(\Gamma)}^\perp}{\vdash \iota(\mathtt{N})^\perp \otimes \iota(A), \overline{\iota(\Gamma)}^\perp, \iota(\mathtt{N})}\ \otimes}{\vdash \iota(\mathtt{N}), \iota(\mathtt{N})^\perp \otimes \iota(A), \overline{\iota(\Gamma)}^\perp}\ exc$$

Note the freshly introduced permutations are cyclic ones thus usable whatever the value of p is, and (ex) is mapped to (ex) thanks to the appropriate parameterized mapping of permutation sets ($\mathfrak{I}_p \subseteq \mathfrak{S}_p$). □

We will now use ι implicitly on formulas since it is mostly the identity on connectives.

It is natural to try to use this translation to transfer results from one system to the other. Let us try to see if one could get cut admissibility for $\mathsf{ILL}(\mathcal{I}, p, c)$ from cut admissibility for $\mathsf{LL}(\iota(\mathcal{I}), p, c, \mathtt{false}, \mathtt{false})$. Assume we have $\Delta \vdash A$ and $\Gamma, A, \Sigma \vdash C$ provable in $\mathsf{ILL}(\mathcal{I}, p, c)$. We then have $\vdash A, \overline{\Delta}^\perp$ and $\vdash C, \overline{\Sigma}^\perp, A^\perp, \overline{\Gamma}^\perp$ provable in $\mathsf{LL}(\iota(\mathcal{I}), p, c, \mathtt{false}, \mathtt{false})$ (Theorem 4). If (cut) is admissible in $\mathsf{LL}(\iota(\mathcal{I}), p, c, \mathtt{false}, \mathtt{false})$ then $\vdash C, \overline{\Sigma}^\perp, \overline{\Delta}^\perp, \overline{\Gamma}^\perp$ is provable. But is it possible to deduce that $\Gamma, \Delta, \Sigma \vdash C$ is provable in $\mathsf{ILL}(\mathcal{I}, p, c)$? This is a problem of conservativity of $\mathsf{LL}(\iota(\mathcal{I}), p, c, \mathtt{false}, \mathtt{false})$ over $\mathsf{ILL}(\mathcal{I}, p, c)$. Unfortunately, there are known counter-examples since the work of H. Schellinx (Schellinx 1991):

- $((X \multimap Y) \multimap 0) \multimap (X \multimap (0 \multimap Y') \multimap Z) \multimap Z$ (due to H. Schellinx)
- $((X \multimap Y) \multimap 0) \multimap (X \otimes \top)$ (due to Y. Lafont)
- $((((0 \multimap 1) \multimap 1) \multimap 0) \multimap 0) \multimap 1$ (due to J.-H. Wu)

These formulas are not provable in $\mathsf{ILL}(\emptyset, p, c)$ but their embedding in $\mathsf{LL}(\emptyset, \mathtt{true}, c, \mathtt{false}, \mathtt{false})$ is provable.

However some sufficient conditions (Schellinx 1991; Laurent 2018) on the considered formulas allow to ensure conservativity. For this we define two particular classes of formulas which are generalizations of "being 0" and of "starting with !". A formula is *almost* 0 if it is of the shape:

$$Z ::= 0 \mid Z \otimes A \mid A \otimes Z \mid Z \,\&\, A \mid A \,\&\, Z \mid Z \oplus Z \mid\, !Z$$

A formula is *almost* ! if it is of the shape:

$$O ::= X \mid 1 \mid 0 \mid O \otimes O \mid O \,\&\, A \mid A \,\&\, O \mid O \oplus O \mid\, !A$$

where, in both cases, A denotes arbitrary formulas.

Theorem 5 (Conservativity [ill_vs_ll.v#ll_to_ill_*_axfree]) ⚠ $\mathcal{I} = \emptyset$ and $p = \mathtt{true}$
If $\vdash C, \overline{\Gamma}^\perp$ *satisfies (at least) one of the following two properties and is provable in* $\mathsf{LL}(\emptyset, \mathtt{true}, c, \mathtt{false}, \mathtt{false})$*:*

- Γ *and* C *do not contain any sub-formula of the shape* $\_ \multimap Z$ *or* $Z \multimap\!\!\circ \_$ *with* Z *an almost* 0 *formula;*

- *all sub-formulas of* Γ *and* C *of the shape* $A \multimap B$, $B \circ\!\!- A$, $\neg A$ *or* $A \circ\!\!-$ *are such that* A *is an almost* ! *formula*

then $\Gamma \vdash C$ *is provable in* $\mathsf{ILL}(\emptyset, \texttt{true}, c)$.

Proof See (Laurent 2018). □

5 Other Linear Systems

5.1 Tensor Logic

Tensor logic TL (Melliès and Tabareau 2010) is a restriction of ILL where implication is replaced by negation. As explained in Section 3.2.2, TL is then a fragment of ILL thanks to the primitive negation in ILL. We give here an alternative presentation (matching the original one) with sequents with at most one formula on the right-hand side.

Given a set `TAtom` (whose elements are denoted X, Y, etc.), *formulas* of propositional tensor logic are defined by the grammar:

$$A ::= X \mid 1 \mid A \otimes A \mid \neg A \mid 0 \mid A \oplus A \mid !A$$

Sequents are of two kinds: either a pair of a list of formulas Γ and of a formula A (*i.e.* $\Gamma \vdash A$), or a list of formulas Γ (*i.e.* $\Gamma \vdash$, sequents with an empty right-hand side). We use the notation $\Gamma \vdash \Pi$ to denote both kinds of sequents, meaning that Π can be either empty or a single formula of TL.

The sequent calculus system $\mathsf{TL}(\mathcal{T}, p, c)$ for tensor logic depends on *three parameters*:

- $\mathcal{T}$ is a set of lists of formulas and of pairs of lists of formulas and formulas (considered as valid sequents in the system);
- p is a Boolean controlling the impact of the exchange rule (thus commutativity);
- c is a Boolean controlling whether or not a primitive *cut* rule belongs to the system.

The *rules of* $\mathsf{TL}(\mathcal{T}, p, c)$ are given in Figure 5. In the (ax_g) rule, by $(\Gamma, \Pi) \in \mathcal{T}$, we mean: either Π is "nothing" and Γ is a list of formulas in $\mathcal{T}$ (thus leading to the sequent $\Gamma \vdash$), or Π is a formula and the pair (Γ, Π) is in $\mathcal{T}$.

If we assume a bijection between `TAtom` and the elements of `IAtom` different from `N` (so that, when $X \in$ `TAtom`, we also write X for its image in `IAtom`, which we know is different from `N`), then $\mathsf{TL}(\mathcal{T}, p, c)$ is simply the fragment of $\mathsf{ILL}(\mathcal{T}, p, c)$ obtained by considering formulas which do not use `N` and rely on the connectives 1, $\otimes$, $\neg$, 0, $\oplus$ and ! only. This requires to see sequents $\Gamma \vdash$ of TL as $\Gamma \vdash \texttt{N}$ in ILL (and conversely).

Theorem 6 (Conservativity of LL over TL [`tl.v#ll_is_tl_axfree`]) ⚠ $\mathcal{T} = \emptyset$ $\Gamma \vdash A$ *(resp.* $\Gamma \vdash$ *) is provable in* $\mathsf{TL}(\emptyset, p, c)$ *if and only if* $\vdash A, \overline{\Gamma}^{\perp}$ *(resp.* $\vdash \texttt{N}, \overline{\Gamma}^{\perp}$*) is provable in* $\mathsf{LL}(\emptyset, p, c, \texttt{false}, \texttt{false})$.

$$\frac{}{X \vdash X}\, ax \qquad ((\Gamma, \Pi) \in \mathcal{T})\ \frac{}{\Gamma \vdash \Pi}\, ax_g \qquad (c)\ \frac{\Delta \vdash A \qquad \Gamma, A, \Sigma \vdash \Pi}{\Gamma, \Delta, \Sigma \vdash \Pi}\, cut$$

$$(\sigma \in \mathfrak{I}_p)\ \frac{\Gamma \vdash \Pi}{\sigma(\Gamma) \vdash \Pi}\, ex \qquad (\sigma \in \mathfrak{S})\ \frac{\Gamma, !\Delta, \Sigma \vdash \Pi}{\Gamma, \sigma(!\Delta), \Sigma \vdash \Pi}\, ex_!$$

$$\frac{}{\vdash 1}\, 1\mathrm{R} \qquad \frac{\Gamma, \Delta \vdash \Pi}{\Gamma, 1, \Delta \vdash \Pi}\, 1\mathrm{L}$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\, \otimes\mathrm{R} \qquad \frac{\Gamma, A, B, \Delta \vdash \Pi}{\Gamma, A \otimes B, \Delta \vdash \Pi}\, \otimes\mathrm{L}$$

$$\frac{A, \Gamma \vdash}{\Gamma \vdash \neg A}\, \neg\mathrm{R} \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash}\, \neg\mathrm{L}$$

$$\frac{}{\Gamma, 0, \Delta \vdash \Pi}\, 0\mathrm{L}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\, \oplus_1\mathrm{R} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\, \oplus_2\mathrm{R} \qquad \frac{\Gamma, A, \Delta \vdash \Pi \qquad \Gamma, B, \Delta \vdash \Pi}{\Gamma, A \oplus B, \Delta \vdash \Pi}\, \oplus\mathrm{L}$$

$$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\, !\mathrm{R} \qquad \frac{\Gamma, A, \Delta \vdash \Pi}{\Gamma, !A, \Delta \vdash \Pi}\, !d\mathrm{L} \qquad \frac{\Gamma, \Delta \vdash \Pi}{\Gamma, !A, \Delta \vdash \Pi}\, !w\mathrm{L}$$

$$\frac{\Gamma, !A, !A, \Delta \vdash \Pi}{\Gamma, !A, \Delta \vdash \Pi}\, !c\mathrm{L}$$

Fig. 5 The $\mathsf{TL}(\mathcal{T}, p, c)$ system.
Π can be either empty or a single formula.

Proof See (Laurent 2018) (it comes from Theorem 5 and from the conservativity of ILL over its fragment TL). □

5.2 Focused Linear Logic

Focusing is a very important property introduced in linear logic. It impacts the analysis of the structure of proofs and it has strong consequences on proof search (Andreoli 1992). We consider here the approaches followed in (Laurent 2004, revised 2017) and (Laurent 2018). We first define two focused systems $\mathsf{LL}_{\mathrm{foc}}$ and $\mathsf{LL}_{\mathrm{Foc}}$ inspired by Girard's system LC (Girard 1991). They correspond to two steps in the building of focused proofs: weak focusing and reversing. Intuitively: "reversing" moves invertible/reversible rules downwards in a proof, while "weak focusing" groups non-invertible rules (as soon as one is applied to the sub-formulas generated by the other).

To make things tractable, we consider here the commutative case only. In the systems $\mathsf{LL}_{\mathrm{foc}}$ and $\mathsf{LL}_{\mathrm{Foc}}$, formulas are formulas of LL but we split them into two

$$\frac{}{\vdash \tilde{X} \mid X}\, ax \qquad (\sigma \in \mathfrak{S})\ \frac{\vdash \Gamma \mid \Pi}{\vdash \sigma(\Gamma) \mid \Pi}\, ex \qquad \frac{\vdash \Gamma \mid A}{\vdash \Gamma, A \mid}\, foc$$

$$\frac{}{\vdash \mid 1}\, 1 \qquad \frac{\vdash \Gamma \mid \Pi}{\vdash \bot, \Gamma \mid \Pi}\, \bot \qquad (\star)\ \frac{}{\vdash \top, \Gamma \mid \Pi}\, \top$$

$$\frac{\vdash \Gamma \parallel A \qquad \vdash \Delta \parallel B}{\vdash \Gamma, \Delta \mid A \otimes B}\, \otimes \qquad \frac{\vdash A, B, \Gamma \mid \Pi}{\vdash A \parr B, \Gamma \mid \Pi}\, \parr$$

$$\frac{\vdash \Gamma \parallel A}{\vdash \Gamma \mid A \oplus B}\, \oplus_1 \qquad \frac{\vdash \Gamma \parallel A}{\vdash \Gamma \mid B \oplus A}\, \oplus_2 \qquad \frac{\vdash A, \Gamma \mid \Pi \qquad \vdash B, \Gamma \mid \Pi}{\vdash A \,\&\, B, \Gamma \mid \Pi}\, \&$$

$$\frac{\vdash ?\Gamma, A \mid}{\vdash ?\Gamma \mid !A}\, ! \qquad \frac{\vdash \Gamma \parallel A}{\vdash ?A, \Gamma \mid}\, ?d \qquad \frac{\vdash \Gamma \mid \Pi}{\vdash ?A, \Gamma \mid \Pi}\, ?w \qquad \frac{\vdash ?A, ?A, \Gamma \mid \Pi}{\vdash ?A, \Gamma \mid \Pi}\, ?c$$

Fig. 6 The LL_{foc} system.
Π can be either empty or a single synchronous formula.
$\vdash \Gamma \parallel A$ is $\vdash \Gamma \mid A$ if A is synchronous and $\vdash \Gamma, A \mid$ if A is asynchronous.
$(\star)$ In the context of a $(\top)$ rule, Γ must contain no formula whose main connective is $\parr$, & or $\bot$.

categories (*synchronous* and *asynchronous* formulas):

$$P ::= X \mid 1 \mid A \otimes A \mid 0 \mid A \oplus A \mid !A$$
$$N ::= \tilde{X} \mid \bot \mid A \parr A \mid \top \mid A \,\&\, A \mid ?A$$

where A, B, etc. denote arbitrary formulas while P denotes a synchronous formula and N denotes an asynchronous formula. Sequents are of two kinds. Either a list of formulas or a list of formulas paired with a *synchronous* formula. We use the notation $\vdash \Gamma \mid \Pi$ to denote both kinds of sequents, meaning that Π can be either empty or a single synchronous formula. We use $\vdash \Gamma \parallel A$ as a notation for $\vdash \Gamma \mid A$ if A is synchronous and $\vdash \Gamma, A \mid$ if A is asynchronous. The *rules of the system* LL_{foc} are presented in Figure 6. The constraint $(\star)$ on the $(\top)$ rule does not impact provability since the more general rule $\dfrac{}{\vdash \top, \Gamma \mid \Pi}\, \top$ is derivable. The constraint $(\star)$ is used to ensure that some proof transformations like reversing do not make the size of the proof increase. We want to avoid for example:

$$\frac{}{\vdash \top, \bot \mid}\, \top \qquad \mapsto \qquad \dfrac{\dfrac{\dfrac{}{\vdash \top \mid}\, \top}{\vdash \bot, \top \mid}\, \bot}{\vdash \top, \bot \mid}\, ex$$

Due to the constraints on the definition of sequents, the formula A in the (*foc*) rule must be synchronous [`llfoc.v#sync_focus`]. The following rule is derivable (A being now an arbitrary formula):

$$\frac{}{\vdash \tilde{X} \mid X}\, ax \qquad (\sigma \in \mathfrak{S})\ \frac{\vdash \Gamma \mid \Pi}{\vdash \sigma(\Gamma) \mid \Pi}\, ex \qquad \frac{\vdash \Gamma \mid A}{\vdash \Gamma, A \mid}\, foc$$

$$\frac{}{\vdash \mid 1}\, 1 \qquad \frac{\vdash \Gamma \mid}{\vdash \bot, \Gamma \mid}\, \bot \qquad (\star)\ \frac{}{\vdash \top, \Gamma \mid}\, \top$$

$$(\star\star)\ \frac{\vdash \Gamma \parallel A \qquad \vdash \Delta \parallel B}{\vdash \Gamma, \Delta \mid A \otimes B}\, \otimes \qquad \frac{\vdash A, B, \Gamma \mid}{\vdash A \parr B, \Gamma \mid}\, \parr$$

$$(\star\star)\ \frac{\vdash \Gamma \parallel A}{\vdash \Gamma \mid A \oplus B}\, \oplus_1 \qquad (\star\star)\ \frac{\vdash \Gamma \parallel A}{\vdash \Gamma \mid B \oplus A}\, \oplus_2 \qquad \frac{\vdash A, \Gamma \mid \qquad \vdash B, \Gamma \mid}{\vdash A \,\&\, B, \Gamma \mid}\, \&$$

$$\frac{\vdash ?\Gamma, A \mid}{\vdash ?\Gamma \mid !A}\, ! \qquad \frac{\vdash \Gamma \parallel A}{\vdash ?A, \Gamma \mid}\, ?d \qquad \frac{\vdash \Gamma \mid}{\vdash ?A, \Gamma \mid}\, ?w \qquad \frac{\vdash ?A, ?A, \Gamma \mid}{\vdash ?A, \Gamma \mid}\, ?c$$

Fig. 7 The $\mathsf{LL}_{\mathrm{Foc}}$ system.
Π can be either empty or a single synchronous formula.
$\vdash \Gamma \parallel A$ is $\vdash \Gamma \mid A$ if A is synchronous and $\vdash \Gamma, A \mid$ if A is asynchronous.
$(\star)$ Γ must contain no formula whose main connective is $\parr$, & or $\bot$.
$(\star\star)$ Γ, Δ must contain no formula whose main connective is $\parr$, &, $\bot$ or $\top$.

$$\frac{\vdash \Gamma \parallel A}{\vdash \Gamma, A \mid}$$

Lemma 17 (Weak focusing [nn_foc.v#weak_focusing] and [llfoc.v#llfoc_to_ll]) $\vdash \Gamma$ *is provable in* LL *if and only if* $\vdash \Gamma \mid$ *is provable in* $\mathsf{LL}_{\mathrm{foc}}$.

Proof The difficult part is to go from LL to $\mathsf{LL}_{\mathrm{foc}}$. The idea used here is to embed proofs from LL into proofs of TL by a $\neg\neg$ translation, then to optimize the translation and to eliminate cuts, and finally to see that, up to some simple rule permutations, the obtained proof in TL is a weakly focused proof if one replaces $\neg A$ with $A^\bot \parr \bot$ and then simplifies it into $A^\bot$ (thanks to the unit isomorphism). Look at (Laurent 2018) for details. □

By applying reversing (*i.e.* by moving invertible/reversible introduction rules downwards), one can get (strong) focusing. The $\mathsf{LL}_{\mathrm{Foc}}$ system is obtained from the $\mathsf{LL}_{\mathrm{foc}}$ system by making all Π empty in the rules of Figure 6 (except in the (*ex*) rule) and by asking the context Γ of sequents $\vdash \Gamma \mid P$ (*i.e.* with a non-empty right-hand side) to contain no formula whose main connective is $\parr$, &, $\bot$ or $\top$, see Figure 7.

Theorem 7 (Focusing [llfoc.v#llfoc_to_llFoc] and [llfoc.v#llFoc_to_ll]) $\vdash \Gamma$ *is provable in* LL *if and only if* $\vdash \Gamma \mid$ *is provable in* $\mathsf{LL}_{\mathrm{Foc}}$.

Proof See (Laurent 2004, revised 2017). □

6 Formalization Ingredients

We discuss here in more details the Rocq formalization itself. It is developed in "vanilla" Rocq with no additional logical axioms. The OLlibs extension of the standard library of Rocq is used: https://github.com/olaure01/ollibs/

It contains material which is mostly independent from Yalla (in particular for list manipulations) and supposed to be potentially integrated in the standard library when mature enough.

6.1 Atoms

Formulas in the logical systems are built on abstract sets of atoms (Atom, IAtom, TAtom). The translations between the systems require functions to move from one set of atoms to another. Sometimes with appropriate properties like injectivity, existence of a right-inverse, etc. Once all such functions and properties are gathered, one should wonder if they are consistent (*i.e.* is it possible to satisfy them all together?).

To keep atom-sets abstract but constraints on them consistent, all the required properties are gathered in the file yalla_ax.v and explicit instances based on nat are provided which ensure consistency (in the spirit of (ANSSI 2021, Developers Rule 4.2)). These explicit values are made opaque to ensure the rest of the library does not depend on the specific instantiation.

In practice, one mostly needs to choose sets of atoms to be denumerable sets with a decidable equality. This is in particular enough to be able to get a formalization in vanilla Rocq with no additional assumptions/axioms, thus avoiding most consistency questions.

6.2 Prop *vs* Type [addendum/proptype.v#]

Most of the deep embeddings of proof systems in Rocq are given as inductive types with output in Prop (see Section 7). For a proof system acting on sequents as lists of formulas, we thus get `proofP : list formula -> Prop`. This means that it represents the *provability predicate* rather than proofs. Representing proof objects themselves (rather than provability) corresponds to choosing Type as output: `proofT : list formula -> Type`.

An immediate consequence is the ability to compute the size of a proof (`size : forall l, proofT l -> nat`). In contrast, this is not possible from `proofP` since it would require to extract computational information from Prop to Type. A standard trick to deal with proof sizes in a Prop setting is to instrument `ProofP` with a parameter in nat to get `proofP_size : nat -> list formula -> Prop` such that `proofP_size n l` corresponds to provability of `l` with a proof of size (at most) `n`. We can then prove `(exists n, proofP_size n l) <-> proofP l` to hide sizes when they are useless. This creates somehow two copies of `proofP` and the same trick has to be used each time we want to "compute" something from proofs, making the formalization highly redundant.

More generally, if you want to do proof transformations and/or to extract information from proofs (typically in the spirit of the Curry-Howard-Lambek correspondence or for computing the interpretation of a proof in a denotational model), you need them

to be in Type. This is why we made the choice of Type as output. The purpose of this library is to formalize the proof theory of linear logic and thus to manipulate proofs rather than provability. For example, cut admissibility [ll_cut.v#cut_admissible] provides a function turning a proof into a cut-free proof of the same sequent.

Similarly, living in Type, one could formalize the Curry-Howard correspondence as an extraction mechanism from proofs to programs. For example, a function from proofs of $\vdash 1 \oplus 1$ (the additive encoding of Booleans) in LL to bool can be defined [addendum/extraction.v#boolean_extraction], or simarly with multiplicative Booleans $\vdash X \otimes X, X^\perp, X^\perp$ where the analysis of permutations is crucial [addendum/extraction.v#multiplicative_boolean_extraction_perm].

6.3 Exchange Rules and Permutations

Due to the absence of weakening and contraction, it is quite common to present the sequent calculus of linear logic as dealing with sequents defined as finite multisets of formulas. By leading to an implicit exchange rule, it makes the system simpler to define. However, this is loosing too much information about the content of proofs, as explained in the introduction. Indeed, not being able to trace occurrences of formulas in a proof (a consequence of the use of multisets) leads to *proof irrelevance*: all proofs of a given formula are equated up to cut elimination. Consider the following proof:

$$\dfrac{\dfrac{\pi_1}{\vdash A}}{\vdash\, !A}\,! \qquad \dfrac{\dfrac{\pi_2}{\vdash A}}{\vdash\, !A}\,! \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,ax}{\vdash\, ?A^\perp, A}\,?d}{\vdash\, ?A^\perp, ?A^\perp, A}\,?w}{\vdash\, ?A^\perp, ?A^\perp, A}\,ex \qquad \dfrac{\vdash\, ?A^\perp, ?A^\perp, A}{\vdash\, ?A^\perp \parr ?A^\perp, A}\,\parr$$

$$\dfrac{\vdash\, !A \otimes !A}{\vdash A}\,\otimes \qquad\qquad \dfrac{\vdash\, ?A^\perp \parr ?A^\perp, A}{\vdash A}\,cut$$

depending whether the permutation used in the (ex) rule commutes the two occurrences of $?A$ or not, the whole proof reduces (by cut elimination) to either π_1 or π_2. This means that, by identifying proofs up to (exponential) exchange rules, one makes all proofs of each formula equal up to cut elimination.

For this reason, we choose to formalize sequents of the various systems with lists of formulas together with an explicit exchange rule dealing with permutations (see also Section 2.2.1).

Now in Rocq, if we use permutations from the standard library (Permutation: forall A : Type, list A -> list A -> Prop), extracting information from a proof cannot depend on the permutations used inside it (again because of the impossibility to eliminate something in Prop to build something in Type). For this reason we had to redefine the inductive type Permutation in Type, leading to Permutation_Type: forall A : Type, list A -> list A -> Type (this is done in OLlibs).

We decided to stick to the notion of permutation defined in the standard library (compositions of adjacent transpositions). This is not a very extensional representation of permutations since there are many ways to represent a given permutation (and

thus many ways to represent each exchange rule of a given proof from the sequent calculus). One could think about a more extensional representation of permutations to be substituted to `Permutation` / `Permutation_Type`. Note that, however, it would have a minor impact on the results formalized in the library since the properties of permutations we rely on do not really depend on the exact way they are represented in `Rocq`.

Another source of non canonicity in the formalization of proofs comes from the exchange rules themselves. One can always introduce an exchange rule using the identity permutation, or collapse two consecutive exchange rules into a single one by composing the underlying permutations. This suggests thinking about alternative ways of dealing with permutations.

To deal with cyclic linear logic, we need to rely on cyclic permutations (a.k.a. circular shifts). They have been introduced in the standard library of `Rocq` in `rocq#12031`[3]. Their analogue with output in `Type` is `CPermutation_Type`. The formalization of the exchange rule of linear logic is then based on `PCPermutation_Type b` (with `b : bool`) which is `Permutation_Type` if `b` is `true` and `CPermutation_Type` if `b` is `false`. And similarly, for intuitionistic linear logic, we use `PEPermutation_Type b` which is `Permutation_Type` if `b` is `true` and equality if `b` is `false`. These parameterized sets of permutations are defined in `ollibs/GPermutation_Type.v`.

6.4 Parameterized Fragments

To deal with parameterized systems, the inductive types defining proofs (see Figure 8 for example) take a tuple (the *fragment*) as argument to specify the values of the parameters. For $\mathtt{LL}(\mathcal{A}, p, c, m_0, m_2)$, a *fragment* is a record containing four Booleans corresponding to p, c, m_0 and m_2, and axioms $\mathcal{A}$ formalized as a dependent pair of a type and a function from it to sequents (*i.e.* of type `{ ptypgax : Type & ptypgax -> list formula }`: indexed families of sequents). This choice for axiom sets makes it easy to deal with unions of axiom sets for example, or to apply to each axiom a transformation given as a function. It is also consistent with the idea of injecting a category in the axiom rules (see (Lambek and Scott 1988, Part I, Sections 1–4)): `ptypgax` is then the set of arrows and the function `ptypgax -> list formula` provides their type.

Alternative possibilities would be to use lists of sequents but it is restricted to finite sets of axioms, or to use functions from sequents to Booleans (characteristic functions) but it is less versatile for example to move from $\mathcal{A}$ to $\{\Gamma \Rightarrow \Delta \mid \Gamma \in \mathcal{A}\}$ (for a given Δ) (see Lemma 9).

Given a type `A`, the representation `{ ptypgax : Type & ptypgax -> A }` is suitable when one wants to (post)compose with a function `A -> B`. In the opposite, the representation `A -> bool` is better suited for (pre)composition with a function `B -> A`. It happens that, in our setting, the first case occurs much more often than the second one.

[3] https://github.com/rocq-prover/rocq/pull/12031

```
Record pfrag := mk_pfrag {
  pcut : bool ;
  pgax : { ptypgax : Type & ptypgax -> list formula } ;
  pmix0 : bool ;
  pmix2 : bool ;
  pperm : bool }.


Inductive ll P : list formula -> Type :=
| ax_r : forall X, ll P (covar X :: var X :: nil)
| ex_r : forall l1 l2, ll P l1 -> PCPermutation_Type (pperm P) l1 l2 ->
                        ll P l2
| ex_wn_r : forall l1 lw lw' l2, ll P (l1 ++ map wn lw ++ l2) ->
              Permutation_Type lw lw' -> ll P (l1 ++ map wn lw' ++ l2)
| mix0_r {f : pmix0 P = true} : ll P nil
| mix2_r {f : pmix2 P = true} : forall l1 l2, ll P l1 -> ll P l2 ->
                                                ll P (l2 ++ l1)
| one_r : ll P (one :: nil)
| bot_r : forall l, ll P l -> ll P (bot :: l)
| tens_r : forall A B l1 l2, ll P (A :: l1) -> ll P (B :: l2) ->
                              ll P (tens A B :: l2 ++ l1)
| parr_r : forall A B l, ll P (A :: B :: l) -> ll P (parr A B :: l)
| top_r : forall l, ll P (top :: l)
| plus_r1 : forall A B l, ll P (A :: l) -> ll P (aplus A B :: l)
| plus_r2 : forall A B l, ll P (A :: l) -> ll P (aplus B A :: l)
| with_r : forall A B l, ll P (A :: l) -> ll P (B :: l) ->
                          ll P (awith A B :: l)
| oc_r : forall A l, ll P (A :: map wn l) -> ll P (oc A :: map wn l)
| de_r : forall A l, ll P (A :: l) -> ll P (wn A :: l)
| wk_r : forall A l, ll P l -> ll P (wn A :: l)
| co_r : forall A l, ll P (wn A :: wn A :: l) -> ll P (wn A :: l)
| cut_r {f : pcut P = true} : forall A l1 l2,
   ll P (dual A :: l1) -> ll P (A :: l2) -> ll P (l2 ++ l1)
| gax_r : forall a, ll P (projT2 (pgax P) a).
```

Fig. 8 The inductive type `ll`.

6.5 Kernel

A major question with proof formalization is: does the formalized result really match its paper version? (see (Pollack 1998; Adams 2016) for example). If, in our case, most statements are simple and do not rely on complex auxiliary definitions, they all build on the formalization of various proof systems and then the crucial questions are: are the systems represented really equivalent to the paper ones? is the inductive type `ll` (Figure 8) really representing LL? The introduction of parameters in particular makes this inductive type not very easy to read. For these reasons, we have decided to apply

a kind of *de Bruijn criterion*: to make possible for a reader to only trust a restricted kernel part of the formalization.

This is the purpose of the files `microyalla/ll.v` and `microyalla/ill.v`. Let us focus on the LL case, the ILL case being very similar. The self-contained `microyalla/ll.v` file provides (in 50 lines of code) a definition of the usual sequent calculus of linear logic. It relies on the standard library for the definition of lists (and `List.map`). In order not to depend on `OLlibs`, it contains a copy of the `Permutation_Type` inductive type. Then the inductive type `ll` defined in `microyalla/ll.v` is designed to match faithfully the paper definition of the sequent calculus of linear logic. These are the 15 lines of code the doubtful reader must look at very carefully.

The purpose of `ll_smp.v` (resp. `ill_smp.v`) is to prove this kernel definition to be equivalent to the inductive type `ll` (from `ll_def.v`) used in the development of the library. More precisely the equivalence with the instance LL($\emptyset$, `true`, `false`, `false`, `false`) is proved. As a consequence, the trust base for the proof of cut admissibility for standard linear logic for example is reduced to `microyalla/ll.v` plus the 2-lines statement [`ll_smp.v#cut_r`] (and the definition of `dual`).

One can even go one step further with this example of cut admissibility by integrating all ingredients in one robust file `addendum/microyalla_cut.v` avoiding external imports except for lists (in the spirit of https://github.com/math-comp/odd-order/blob/master/theories/stripped_odd_order_theorem.v).

6.6 Templates

Due to the introduction of parameters, the inductive types defining logical systems (LL, ILL, TL, etc.) are uselessly complex for most concrete applications. The recommended approach, when working on a given variant $\mathcal{V}$ of linear logic, is to define an ad hoc inductive type so that using it could be as natural as possible. To take benefits from the `Yalla` library, the idea is to prove an equivalence with (a fragment of) one of the systems already defined in the library and then to inherit from it the results about $\mathcal{V}$. Various template files illustrating this approach are distributed with `Yalla`. We describe here in each case the variant of LL under consideration and the associated properties which are formalized:

- `ll_smp.v`: *"Standard" linear logic* (LL) is defined directly from the `microyalla/ll.v` kernel (Section 6.5) and proved equivalent to LL($\emptyset$, `true`, `false`, `false`, `false`). Axiom expansion and cut elimination are inherited from the library.
- `ill_smp.v`: *"Standard" intuitionistic linear logic* (ILL) is defined directly from the `microyalla/ill.v` kernel (Section 6.5) and proved equivalent to ILL($\emptyset$, `true`, `false`). Axiom expansion and cut elimination are inherited from the library.
- `mell2.v`: The *unit-free multiplicative-exponential fragment of linear logic (*MELL*) extended with the (mix) rule* is defined and proved equivalent to

$\mathsf{LL}(\emptyset, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{true})$ if one uses formulas built with the connectives $\otimes$, $\parr$, ! and ? only. Axiom expansion and cut elimination are inherited from the library.

- `lambek.v`: A variant of the *Lambek calculus* (Lambek 1958) with cartesian (*i.e.* additive) product is defined and proved equivalent to $\mathsf{ILL}(\emptyset, \texttt{false}, \texttt{false})$ (a non-commutative fragment) if one uses formulas built with the connectives $\multimap$, & and $\top$ only. Axiom expansion and cut elimination are inherited from the library.
- `llpol.v`: *Polarized linear logic* LL_{pol} (Laurent 2002) is defined and proved equivalent to $\mathsf{LL}(\emptyset, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{false})$ if one uses *polarized formulas* only:

$$P ::= X \mid 1 \mid P \otimes P \mid 0 \mid P \oplus P \mid !N$$
$$N ::= \tilde{X} \mid \bot \mid N \parr N \mid \top \mid N \,\&\, N \mid ?P$$

  Axiom expansion and cut elimination are inherited from the library. Specific results are then derived. Mostly the fact that sequents made of negative formulas together with at most one positive formula (*polarized sequents*) which are provable in LL are exactly those provable by an LL_{pol} proof containing polarized sequents only.
- `mell_Prop.v`: The *unit-free multiplicative-exponential fragment of linear logic (*MELL*)* is defined but as an inductive type in `Prop` (rather than `Type`). It is proved equivalent to existence of a proof in $\mathsf{LL}(\emptyset, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{false})$ if one uses formulas built with the connectives $\otimes$, $\parr$, ! and ? only, through the use of the `inhabited` embedding of `Type` into `Prop`. Axiom expansion and cut elimination can also be inherited from the library in this `Prop` setting. The converse approach (building results in `Prop` in the library and trying to use them in `Type`) would clearly not be possible.

All these template files follow mostly the same pattern (which is the one suggested to the users). Let us describe it in the particular case of `mell2.v`:

- find in the library the system that contains yours (here `ll_def.ll`);
- define an inductive type `formula` for your formulas (often relying on appropriate atom sets from `Yalla`, here `formulas.Atom`);
- define an (usually injective) embedding `mell2ll` from the new formulas to those in the associated `Yalla` system;
- prove compatibility with formula operations (`mell2ll_dual`, `mell2ll_map_wn` and `mell2ll_map_wn_inv`);
- define the proof system (`mell`) with rules written the way they should be (*i.e.* the way you want them to be, not the way they could be/are in `Yalla`);
- characterize the appropriate associated instance in `Yalla` (`pfrag_mell`);
- prove equivalence between the local system and the `Yalla` instance (`mell2mellfrag` and `mellfrag2mell`);
- thanks to this equivalence, import results from `Yalla` (for example `ax_gen_r` and `cut_r`).

Another example is presented in details in Appendix D.

6.7 Multisets-Based Systems

We have explained our choice to work in `Type` (Section 6.2) and with explicit exchange rules (Section 6.3). However this may not be the most natural choice for users working with commutative systems and not concerned with proof relevance. For this reason we also propose variants in `Prop` dealing with a representation of sequents as finite multisets.

The idea is to rely on a notion of finite multisets satisfying the following properties:

- it is possible to extract a list of *elements* from a finite multiset;
- *equality* of finite multisets corresponds to *permutation* of their elements.

This is given through an appropriate axiomatization in `OLlibs` together with an explicit construction satisfying these axioms. More precisely we propose two approaches following what can be found in the literature. Either a notion of finite multiset whose equality is `Rocq` Leibniz equality, or a setoid approach where equality of finite multisets is given by an equivalence relation.

We provide two representations of the *multiplicative-exponential fragment of linear logic (*MELL*)*:

- `mell_mset.v`: List operations are replaced with multiset operations. The exchange rule becomes implicit since equality of multisets corresponds to equality of lists up to permutation.
- `mell_msetoid.v`: Sequents are considered up to an equivalence relation `meq`, with an explicit exchange rule giving compatibility of provability with this relation. This is not so different from previous systems with explicit exchange rules but `meq` has no computational content (the underlying permutations are lost).

7 Related Works

As reflected by the `Yalla` acronym (see title of the paper), this project is far from being the first one to address the formalization of linear logic in `Rocq`. There is ongoing work on gathering such formalizations in proof assistants through the FormalizedLLBib project[4]. Collected formalizations in `Rocq` are maintained to compile with up-to-date `Rocq` versions.

We focus here on the comparison between `Yalla` and other `Rocq` formalizations which can be summed up in the following table:

[4] https://github.com/ComputerAidedLL/FormalizedLLBib

	Prop *vs* Type	seq	(I)LL	qtf	a-exp	c-adm	focus
LL in Coq[5]	Prop	lists	ILL				
ll-coq[6]	Prop	lists	LL				
ILL Narratives[7]	Prop	fmsets	ILL				
LinearLogic[8]	Prop	lists	ILL				
Linear-logic[9]	Prop	fmsets	ILL				
AUGER_LinearLogic[10]	Prop	lists	LL		✓		
coq-ll[11]	Prop	fmsets	LL	✓	✓	✓	✓
Yalla	Type	lists[12]	both		✓	✓	✓

The first four columns are about the core choices for the systems and the representation of sequents and proofs:

- "Prop *vs* Type": the output sort of the inductive type defining proofs;
- "seq": the structure used for the representation of sequents (fmsets = finite multi-sets, lists = lists);
- "(I)LL": the system under consideration can be classical linear logic (LL) or intuitionistic linear logic (ILL), or both;
- "qtf": formulas and proofs include the representation of quantifiers (otherwise, propositional only).

We then consider the various results/components considered in the formalizations, and which are all mostly contained in the following list:

- "a-exp": proof of the axiom-expansion property (Lemma 3);
- "c-adm": proof of cut admissibility (Theorem 1);
- "focus": definition of a focused system with the proofs of soundness and completeness of such a system (Theorem 7).

Outside Rocq, we already talked about (Chaudhuri et al. 2017) in the introduction (Section 1). It is a formalization in the Abella proof system. Let us give its key ingredients (as given in the extended version (Chaudhuri et al. 2019)) in the spirit of the table above: fmsets, LL, qtf, c-adm, focus (there is no point about Prop/Type distinction in Abella), *i.e.* same components as in coq-ll (Xavier et al. 2017).

Except for the representation of quantifiers, Yalla appears currently as the richer corpus of formalized results about linear logic, dealing with both LL and ILL and providing properties relating them. Various systems and syntactic results not mentioned in the table only occur in Yalla (non-commutative variants, mix rules, deduction lemma, sub-formula property, tensor logic TL, polarization, Lambek calculus, etc.). Moreover,

[5] https://github.com/ComputerAidedLL/PowerWebster_ILL (Power and Webster 1999)
[6] https://github.com/ppedrot/ll-coq
[7] https://github.com/Matafou/ill_narratives (Bosser et al. 2011)
[8] https://github.com/acowley/LinearLogic (Cowley and Taylor 2011)
[9] https://github.com/kai-qu/linear-logic
[10] https://github.com/ComputerAidedLL/Auger_LinearLogic
[11] https://github.com/meta-logic/coq-ll (Xavier et al. 2017) and https://github.com/meta-logic/coq-fll (Felty et al. 2021)
[12] With support for finite multi-sets (Section 6.7).

as opposed to the other projects presented above, it is designed as an evolving library oriented towards usability for future formalization projects.

8 Future Works

Current ongoing work includes turning c (the Boolean deciding for the availability of the (*cut*) rule) into a more refined criterion for selecting the set of formulas on which the (*cut*) rule is accepted. We also work on generalizing the management of (*mix*) rules by considering n-ary such rules. Alternative approaches to focusing based on Andreoli's triadic system (Andreoli 1992) (formalized in `coq-ll`[11]) are tested. Phase semantics in the spirit of Larchey's work[13] (also present in `ll-coq`[6]) is under consideration.

The main new direction for extending the scope of the `Yalla` library is to incorporate quantifiers. This is already done for example in (Xavier et al. 2017) by using parametric higher-order abstract syntax. We would like to rely on simpler structures to avoid frightening users not familiar with the formalization of binders. Our current plan is to apply the anti-locally-nameless approach (Laurent 2021).

The structure of exponential connectives should be decomposed to incorporate variations on the validated logical principles so that systems like light logics (Girard 1998) could be covered. We plan to rely on super exponentials (Bauer and Laurent 2021) for that. This would also be the opportunity to take into account the third level of commutation constraints on the exponential connectives used in (Yetter 1990) and discussed in Appendix A.

On more technical aspects, support for automated solving of permutation constraints should be developed. In particular when the choice of a permutation does not matter or when the solution happens to be unique. Tools for proving non-provability results are also interesting to investigate since currently such properties are rather tedious to settle (see for example non-conservativity proofs [`ill_vs_ll.v#Non_Conservativity`]).

Finally we want to be able to deal with proof transformations at a finer level than currently done for cut admissibility. Defining local rule permutations is currently made difficult by the presence of exchange rules. We would like to get a better control on them by going towards a more "implicit" treatment of exchange rules which would be possible by building rules in such a way that exchange becomes an (mostly) admissible rule. This seems to be a necessary step for considering seriously denotational semantics and in particular for proving invariance of denotational models through small cut-elimination steps in proofs. It would also be necessary for making possible the formalization of the sequent-calculus part of (Di Guardia and Laurent 2025) (including confluence up to rule permutations of the sequent calculus of multiplicative additive linear logic), a major testing project for future `Yalla` versions.

Acknowledgements. I would like to thank Damien Pous for his many suggestions during the development of this work, as well as Cyril Cohen for his fruitful comments. During the building process of the `Yalla` library, many discussions with students had an important impact on the design choices. Let me name Christophe Lucas, Esaïe Bauer, Mathias Michel, Étienne Calliès and Rémi Di Guardia. I also want to thank

[13] https://github.com/DmxLarchey/Coq-Phase-Semantics

the anonymous reviewers who made important suggestions to make this paper easier to read.

Declarations.

- Funding: This work was supported by the ANR projet RECIPROG (ANR-21-CE48-0019) operated by the French National Research Agency (ANR). This work was also supported by IRN Linear Logic and IRN Logic and Interaction.
- Author contribution: The manuscript was entirely written by O.L.

Appendix A Strong Exponential Exchange Rule

While the rule ($ex_?$) says that ?-formulas commute with each other, there is an alternative stronger variant where a ?-formula is allowed to commute with any other formula (Yetter 1990). It stands between the cyclic system presented here and the fully commutative one. This variant is not taken into account currently in the `Yalla` library, but let us just make a few comments here about the induced provable sequents.

Lemma 18 (Strong Exponential Exchange Rules [addendum/expexchange.v#StrongExponentialExchange])
The following (singleton) sets of rules are inter-derivable over $\mathtt{LL}(\mathcal{A}, \mathtt{false}, c, m_0, m_2)\setminus\{ex_?\}$*:*

$$\left\{\dfrac{\vdash \Gamma, B, ?A, \Sigma}{\vdash \Gamma, ?A, B, \Sigma}\, ex_?{}^l_s\right\} \quad \left\{\dfrac{\vdash \Gamma, \Delta, ?A, \Sigma}{\vdash \Gamma, ?A, \Delta, \Sigma}\, ex_?{}^l_{se}\right\} \quad \left\{\dfrac{\vdash ?A, \Gamma, ?A, \Delta}{\vdash ?A, \Gamma, \Delta}\, ?c^l_s\right\}$$

$$\left\{\dfrac{\vdash \Gamma, ?A, B, \Sigma}{\vdash \Gamma, B, ?A, \Sigma}\, ex_?{}^r_s\right\} \quad \left\{\dfrac{\vdash \Gamma, ?A, \Delta, \Sigma}{\vdash \Gamma, \Delta, ?A, \Sigma}\, ex_?{}^r_{se}\right\} \quad \left\{\dfrac{\vdash ?A, \Gamma, ?A, \Delta}{\vdash \Gamma, ?A, \Delta}\, ?c^r_s\right\}$$

Proof Here are some of the key inter-derivability proofs:

$$\dfrac{\dfrac{\vdash ?A, \Gamma, ?A, \Delta}{\vdash ?A, ?A, \Gamma, \Delta}\, ex_?{}^l_{se}}{\vdash ?A, \Gamma, \Delta}\, ?c \qquad \dfrac{\dfrac{\dfrac{\vdash \Gamma, A, ?B, \Delta}{\vdash A, ?B, \Delta, \Gamma}\, exc}{\vdash ?B, A, ?B, \Delta, \Gamma}\, ?w}{\vdash ?B, A, \Delta, \Gamma}\, ?c^l_s}{\vdash \Gamma, ?B, A, \Delta}\, exc \qquad \dfrac{\dfrac{\dfrac{\vdash ?A, \Gamma, ?A, \Delta}{\vdash ?A, \Delta, ?A, \Gamma}\, exc}{\vdash ?A, \Delta, \Gamma}\, ?c^l_s}{\vdash \Gamma, ?A, \Delta}\, exc$$

□

Example. In order to justify that this second family of exponential exchange rules (Lemma 18) is strictly stronger than the first one (Lemma 5), we can look at the sequent $\vdash X, Y, ?(\tilde{X} \otimes \tilde{Y})$.

This sequent is provable using the ($ex_?{}^r_s$) rule [addendum/expexchange.v#strongcommut]:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \tilde{X}, X}\, ax \quad \dfrac{}{\vdash \tilde{Y}, Y}\, ax}{\vdash \tilde{X} \otimes \tilde{Y}, Y, X}\, \otimes}{\vdash ?(\tilde{X} \otimes \tilde{Y}), Y, X}\, ?d}{\vdash X, ?(\tilde{X} \otimes \tilde{Y}), Y}\, exc}{\vdash X, Y, ?(\tilde{X} \otimes \tilde{Y})}\, ex_?{}^r_s$$

However, if $X \neq Y$, the sequent $\vdash X, Y, ?(\tilde{X} \otimes \tilde{Y})$ is not provable in the weaker system [addendum/expexchange.v#weakcommut]. By looking at all possible last rules, we can prove that any tentative proof would contain a branch with a sequent in which the occurrences of variables (without taking duality into account) are a cyclic permutation of $X, Y, X, Y, \ldots, X, Y$ or $X, Y, X, Y, \ldots, X, Y, X$ or $Y, X, Y, X, \ldots, Y, X, Y$ without ever being provable (because of the persistent interleaving of Xs and Ys).

Appendix B Mix Rules

One could wonder why there is a kind of duality between $1 \vdash \bot$ (in Lemma 12) and $\bot \vdash 1$ (Lemma 13), while $A \otimes B \vdash A \parr B$ appears in Lemma 13 but nothing about $A \parr B \vdash A \otimes B$ occurs in Lemma 12. Remember that 1 is a unit for $\otimes$ and $\bot$ is a unit for $\parr$. A key principle of multiplicative linear logic is $A \otimes (B \parr C) \vdash (A \otimes B) \parr C$:

$$\dfrac{\dfrac{}{\vdash A, A^\perp}\, ax_e \qquad \dfrac{\dfrac{\dfrac{}{\vdash C^\perp, C}\, ax_e \qquad \dfrac{}{\vdash B^\perp, B}\, ax_e}{\vdash C^\perp \otimes B^\perp, B, C}\, \otimes}{\vdash B, C, C^\perp \otimes B^\perp}\, exc}{\dfrac{\dfrac{\dfrac{\vdash A \otimes B, C, C^\perp \otimes B^\perp, A^\perp}{\vdash (A \otimes B) \parr C, C^\perp \otimes B^\perp, A^\perp}\, \parr}{\vdash C^\perp \otimes B^\perp, A^\perp, (A \otimes B) \parr C}\, exc}{\vdash (C^\perp \otimes B^\perp) \parr A^\perp, (A \otimes B) \parr C}\, \parr}\, \otimes$$

When $\bot \vdash 1$ is provable, we can adapt it into:

$$\dfrac{\dfrac{}{\vdash A, A^\perp}\, ax_e \qquad \dfrac{\dfrac{\dfrac{}{\vdash B^\perp, B}\, ax_e \qquad \dfrac{}{\vdash 1, 1}}{\vdash B^\perp \otimes 1, 1, B}\, \otimes}{\vdash 1, B, B^\perp \otimes 1}\, exc}{\dfrac{\dfrac{\dfrac{\vdash A \otimes 1, B, B^\perp \otimes 1, A^\perp}{\vdash (A \otimes 1) \parr B, B^\perp \otimes 1, A^\perp}\, \parr}{\vdash B^\perp \otimes 1, A^\perp, (A \otimes 1) \parr B}\, exc}{\vdash (B^\perp \otimes 1) \parr A^\perp, (A \otimes 1) \parr B}\, \parr}\, \otimes$$

that is (up to unit isomorphisms): $A \otimes B \vdash A \parr B$. Conversely, if we instantiate $A \otimes B \vdash A \parr B$ with $A := 1$ and $B := \bot$, one gets $1 \otimes \bot \vdash 1 \parr \bot$, thus $\bot \vdash 1$ (up to unit isomorphisms). However the principle $A \parr B \vdash A \otimes B$ is of a very different nature, with no relation to $1 \vdash \bot$. For a more geometric interpretation, $A \parr B \vdash A \otimes B$ corresponds to a cycle in proof-nets, while $1 \vdash \bot$ is about validating the empty proof-net. $A \otimes B \vdash A \parr B$ and $\bot \vdash 1$ on their side are related with disconnectedness in proof-nets. The loss of "duality" thus comes from the fact that there is no provable "dual" principle associated to $A \otimes (B \parr C) \vdash (A \otimes B) \parr C$.

Appendix C Example: Consistency with (mix_0)

We present here a formalization exercise. It may look as (and mostly is) a curiosity, but it happened to be useful in the design of a system in some recent work (Laurent 2026).

Exercise. In the unit-free fragment of classical multiplicative linear logic with both (mix) and (mix_0), called unit-free $\mathsf{MLL}_{0,2}$, there is no formula A such that both $\vdash A$ and $\vdash A^\perp$ are provable.

In we replace unit-free $\mathsf{MLL}_{0,2}$ by some sub-system S of linear logic, the standard proof for this result is by contradiction with the following steps:

- prove cut-admissibility in S;
- deduce a proof of the empty sequent $\vdash$ from the proofs of $\vdash A$ and $\vdash A^\perp$ by cut admissibility;
- conclude with a contradiction since $\vdash$ is not provable in S.

However this approach is not possible in unit-free $\mathsf{MLL}_{0,2}$ since $\vdash$ *is* provable thanks to the (mix_0) rule. Note also that the result becomes false if we add the multiplicative units 1 and $\bot$ since then we have:

$$\frac{}{\vdash 1}\,1 \qquad \text{and} \qquad \frac{\dfrac{}{\vdash}\,mix_0}{\vdash \bot}\,\bot$$

The trick we are going to use is to apply the above proof blueprint to LL_2: classical linear logic with (mix) but without (mix_0). Let us go to the formalization. We follow here step by step the `mll02.v` file.

We first import the required modules from `OLlibs` and `Yalla`:

```
From Yalla Require Import List_more Permutation_Type ll_cut.
Import ListNotations.
```

Instead of redefining formulas, we simply rely on the `Yalla` ones:

```
Print formula.
(*
Inductive formula : Set :=
    var : Atom -> formula
  | covar : Atom -> formula
  | one : formula
  | bot : formula
  | tens : formula -> formula -> formula
  | parr : formula -> formula -> formula
  | zero : formula
  | top : formula
  | aplus : formula -> formula -> formula
  | awith : formula -> formula -> formula
  | oc : formula -> formula
```

```
  | wn : formula -> formula.
*)
```

It is then simple to define the rules of unit-free $\mathsf{MLL}_{0,2}$:

```
Inductive mll02 : list formula -> Type :=
| ax A : mll02 [A; dual A]
| ex l1 l2 : mll02 l1 -> Permutation_Type l1 l2 -> mll02 l2
| mix0 : mll02 nil
| mix2 l1 l2 : mll02 l1 -> mll02 l2 -> mll02 (l1 ++ l2)
| ten A B l1 l2 :
   mll02 (A :: l1) -> mll02 (B :: l2) -> mll02 (tens A B :: l1 ++ l2)
| par A B l : mll02 (A :: B :: l) -> mll02 (parr A B :: l).
```

We now look for LL_2 as an instance of $\mathsf{LL}(\mathcal{A}, p, c, m_0, m_2)$: no axiom, no cut, full exchange, (*mix*) but not (mix_0), that is $\mathsf{LL}(\emptyset, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{true})$.

```
Definition ll2_frag := mk_pfrag false NoAxioms false true true.
Definition ll2 := ll ll2_frag.
```

The key property we want to show is that provability in unit-free $\mathsf{MLL}_{0,2}$ entails provability in LL_2, as soon as we restrict to *non-empty* sequents. The proof goes by induction. In order to be able to use the induction hypothesis, we need to prove that:

- being non-empty is preserved by permutation;
- a non-empty concatenation of two lists has at least one non-empty component;
- the (mix_0) rule cannot be used since it introduces an empty sequent.

```
Lemma mix02_to_mix2 l : mll02 l -> l <> [] -> ll2 l.
```

```
Proof. [...] Qed.
```

Now we move to properties of the (mix_0)-free system LL_2 (in particular weak consistency [addendum/consistency.v#not_empty_provable]). We go by contradiction. If $\vdash A$ and $\vdash A^\perp$ were provable in unit-free $\mathsf{MLL}_{0,2}$, as these sequents are not empty, they would also be provable in LL_2. But the cut rule is admissible in LL_2 thus we get a proof of $\vdash$, a contradiction.

```
Require Import consistency.
```

```
Lemma mll02_weak_consistency : notT { A & mll02 [A] & mll02 [dual A] }.
```

```
Proof. [...] Qed.
```

Appendix D Example: Tensorial Logic

Assume you have just discovered the *tensorial logic* presented on the last page of (Melliès 2012) and you want to do some formal development on the characterization of the formulas satisfying $\neg\neg A \vdash A$.

$$\frac{}{A \vdash A}\ ax \qquad \frac{\Gamma \vdash A \qquad \Delta_1, A, \Delta_2 \vdash B}{\Delta_1, \Gamma, \Delta_2 \vdash B}\ cut$$

$$\frac{A, \Gamma \vdash \bot}{\Gamma \vdash \neg A}\ \neg\mathrm{R} \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \bot}\ \neg\mathrm{L}$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ \otimes\mathrm{R} \qquad \frac{\Delta_1, A, B, \Delta_2 \vdash C}{\Delta_1, A \otimes B, \Delta_2 \vdash C}\ \otimes\mathrm{L}$$

$$\frac{}{\vdash 1}\ 1\mathrm{R} \qquad \frac{\Delta_1, \Delta_2 \vdash A}{\Delta_1, 1, \Delta_2 \vdash A}\ 1\mathrm{L}$$

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\ ex$$

Fig. D1 Tensorial Logic (from (Melliès 2012))

This is a toy example ignoring the fact that tensor logic (Section 5.1) is already part of the `Yalla` library with additional parameters and connectives. We follow here step by step the `tl_smp.v` file.

You will rely on some of the tools from `OLlibs` as well as about `ILL` from `Yalla`. This requires to load the appropriate modules:

```
Set Implicit Arguments.
From Yalla Require Import List_more Permutation_Type_more ill_cut.
Import ListNotations.
```

The paper describes formulas by:

$$A ::= X \mid A \otimes A \mid 1 \mid \neg A \mid \bot$$

where X is an atom. Note that we rely on the notations from the paper, $\bot$ has nothing to do with the multiplicative unit of classical linear logic.

The logic appears as a restriction of `ILL` with a particular constant $\bot$ (see rules on Figure D1). By identifying that $\bot$ will correspond to `N`, you will have to map atoms to atoms of `ILL` different from `N`. For this reason you choose [`yalla_ax.v#Atom`] as set of atoms since it will come with an injection [`yalla_ax.v#a2i`] into atoms of `ILL` different from `N`. The definition of *tensorial formulas* then follows the grammar from the paper:

```
Inductive tformula :=
| tvar (_ : yalla_ax.Atom) | ttens (_ _ : tformula) | tone
| tneg (_ : tformula) | tbot.
```

The appropriate choice of atoms helps you defining the embedding of tensorial formulas into `ILL` *formulas*. Connectives are mapped to the corresponding ones, and `N` is used for $\bot$ (as it will occupy the right-hand side of sequents in negation rules):

```
Fixpoint t2i A :=
match A with
| tvar X => ivar (yalla_ax.a2i X)
| ttens A B => itens (t2i A) (t2i B)
| tone => ione
| tneg A => ineg (t2i A)
| tbot => N
end.
```

Let us move to proofs now. The paper mentions that sequents are pairs of a (possibly empty) sequence of formulas and a (unique) formula: $A_1, \ldots, A_n \vdash B$. This leads us to the definition of an *inductive type* with type `list tformula -> tformula -> Type` for proofs. Each constructor exactly matches a rule from the paper (see Figure D1).

```
Inductive tl : list tformula -> tformula -> Type :=
| ax_tr A : tl [A] A
| cut_tr A l0 l1 l2 B :
   tl l0 A -> tl (l1 ++ A :: l2) B -> tl (l1 ++ l0 ++ l2) B
| neg_trr A l : tl (A :: l) tbot -> tl l (tneg A)
| neg_tlr A l : tl l A -> tl (l ++ [tneg A]) tbot
| tens_trr A B l1 l2 : tl l1 A -> tl l2 B -> tl (l1 ++ l2) (ttens A B)
| tens_tlr A B l1 l2 C :
   tl (l1 ++ A :: B :: l2) C -> tl (l1 ++ ttens A B :: l2) C
| one_trr : tl [] tone
| one_tlr l1 l2 A : tl (l1 ++ l2) A -> tl (l1 ++ tone :: l2) A
| ex_tr A B l1 l2 C :
   tl (l1 ++ A :: B :: l2) C -> tl (l1 ++ B :: A :: l2) C.
Arguments cut_tr [_ _ _ _ _] _ _, _ [_ _ _ _] _ _, _ _ _ _ _ _ _ _.
Arguments tens_tlr [_ _ _ _ _] _.
Arguments one_tlr [_ _ _] _.
Arguments ex_tr [_ _ _ _ _] _.
```

Since our goal is to formalize some properties about provability, the presence of the cut rule in the system is blocking. We thus want to prove that this rule is admissible, meaning that we can restrict proofs to cut-free ones without modifying the provability. We start by defining a cut-freeness test for proofs (which outputs the Boolean `true : bool` if the input proof is cut-free):

```
Fixpoint cut_free_tl l A (pi : tl l A) :=
match pi with
| ax_tr _ | one_trr => true
| cut_tr _ _ => false
| neg_trr pi1 | neg_tlr pi1
| tens_tlr pi1 | one_tlr pi1 | ex_tr pi1 => cut_free_tl pi1
| tens_trr pi1 pi2 => andb (cut_free_tl pi1) (cut_free_tl pi2)
end.
```

The way we will obtain cut admissibility is by relying on cut admissibility for ILL already proved in `Yalla`. We first look for the appropriate instance of $\mathsf{ILL}(\mathcal{I}, p, c)$. We

want a system without axioms nor cut but with commutativity: ILL($\emptyset$, `true`, `false`). This particular instance is already defined [ill_cut.v#ill_ll]. The embedding of tensorial logic into ILL goes directly by mapping each rule to each corresponding one. There are three cases which deserve some additional work:

- the axiom rule which is restricted to atoms in `ill_ll`, thus we use [ill_def.v#ax_exp_ill];
- the cut rule is not present in `ill_ll`, but already shown to be admissible [ill_cut.v#cut_ll_ir];
- the exchange rule which is defined with transpositions, while it appears with arbitrary permutations in `ill_ll`.

```
Lemma tl2ill l C : tl l C -> ill_ll (map t2i l) (t2i C).
```

```
Proof. [...] Qed.
```

The reverse direction from ILL($\emptyset$, `true`, `false`) to tensorial logic is more tricky. This is not really surprising since it is the place where we try to move from a big system to a smaller one. Anyway, this is still rather systematic by analysing the structure of the image sequents in `ill_ll`. This is clearly a place where one would be happy to rely on more automation in the future. We choosed `ill_ll` as a cut-free system. In order to be able to rely on this property, we prove that the tensorial proof we build is also cut-free.

```
Lemma ill2tl l C : ill_ll (map t2i l) (t2i C) ->
{ pi : tl l C | cut_free_tl pi = true }.
```

```
Proof. [...] Qed.
```

By simply composing the two results, we get cut admissibility for tensorial logic:

```
Lemma cut_admissibility_tl l A :
  tl l A -> { pi : tl l A | cut_free_tl pi = true }.
```

```
Proof. [...] Qed.
```

Our goal is to prove that formulas satisfying $\neg\neg A \vdash A$ must contain $\neg$ or $\bot$. We first define a Boolean function for characterizing the $\neg$-free $\bot$-free formulas:

```
Fixpoint bot_neg_free A :=
match A with
| tbot | tneg _ => false
| ttens A B => andb (bot_neg_free A) (bot_neg_free B)
| _ => true
end.
```

The proof of the final result then goes by induction on a *cut-free* tensorial logic proof of $\neg\neg A \vdash A$:

```
Lemma neg_neg_elim_is_bot_neg A :
  tl [tneg (tneg A)] A -> bot_neg_free A = false.
```

```
Proof. [...] Qed.
```

We now have a fully formalized proof. The main point has been to identify the appropriate associated system in `Yalla` and to prove the equivalence with tensorial logic.

This has been the key step to inherit cut-admissibility in tensorial logic which was necessary to make our proof by induction on cut-free proofs.

Appendix E Legenda for Some Definitions

Rocq	paper	paper ref	Rocq file
ax_exp	(ax_e)	Lemma 3	yalla/ll_def.v
dcol	$(?c^l)$	Lemma 5	addendum/expexchange.v
dcor	$(?c^r)$	Lemma 5	addendum/expexchange.v
ex	(ex)	Lemma 4	addendum/exchange.v
exc	(exc)	Lemma 2	addendum/exchange.v
excl	(exc_1^l)	Lemma 2	addendum/exchange.v
excr	(exc_1^r)	Lemma 2	addendum/exchange.v
ext	(ex_t)	Lemma 4	addendum/exchange.v
extl	(ex_t^l)	Lemma 4	addendum/exchange.v
exwn	$(ex_?)$	Lemma 5	addendum/expexchange.v
mco	$(?c_e)$	Lemma 5	addendum/expexchange.v
sdcol	$(?c_s^l)$	Lemma 18	addendum/expexchange.v
sdcor	$(?c_s^r)$	Lemma 18	addendum/expexchange.v
sdexwnl	$(ex_{?\,se}^{\ l})$	Lemma 18	addendum/expexchange.v
sdexwnr	$(ex_{?\,se}^{\ r})$	Lemma 18	addendum/expexchange.v
sexwnl	$(ex_{?\,s}^{\ l})$	Lemma 18	addendum/expexchange.v
sexwnr	$(ex_{?\,s}^{\ r})$	Lemma 18	addendum/expexchange.v

References

Abrusci M (1991) Phase semantics and sequent calculus for pure noncommutative classical linear propositional logic. Journal of Symbolic Logic 56(4):1403–1451. https://doi.org/10.2307/2275485

Abrusci M, Maringelli E (1998) A new correctness criterion for cyclic proof nets. Journal of Logic Language and Information 7:449–459. https://doi.org/10.1023/A:1008354130493

Adams M (2016) Proof auditing formalised mathematics. Journal of Formalized Reasoning 9(1):3–32. https://doi.org/10.6092/issn.1972-5787/4576

Andreoli JM (1992) Logic programming with focusing proofs in linear logic. Journal of Logic and Computation 2(3):297–347. https://doi.org/10.1093/logcom/2.3.297

ANSSI FNCA (2021) Requirements on the use of Coq in the context of common criteria evaluations. Tech. Rep. v1.1, French National Cybersecurity Agency (ANSSI) and INRIA, URL https://cyber.gouv.fr/sites/default/files/document/anssi-requirements-on-the-use-of-coq-in-the-context-of-common-criteria-evaluations-v1.1-en.pdf

Avron A (1988) The semantics and proof theory of linear logic. Theoretical Computer Science 57(2–3):161–184. https://doi.org/10.1016/0304-3975(88)90037-0

Baillot P, Das A (2016) Free-cut elimination in linear logic and an application to a feasible arithmetic. In: Talbot JM, Regnier L (eds) 25th EACSL Annual Conference on Computer Science Logic, CSL, Leibniz International Proceedings in Informatics (LIPIcs), vol 62. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, pp 40:1–40:18, https://doi.org/10.4230/LIPIcs.CSL.2016.40

Bauer E, Laurent O (2021) Super exponentials in linear logic. In: Dal Lago U, de Paiva V (eds) Proceedings Second Joint International Workshop on Linearity and Trends in Linear Logic and Applications 2020, pp 50–73, https://doi.org/10.4204/EPTCS.353.3

Bellin G (1997) Subnets of proof-nets in multiplicative linear logic with MIX. Mathematical Structures in Computer Science 7(6):663–669. https://doi.org/10.1017/S0960129597002326

Bosser AG, Courtieu P, Forest J, et al (2011) Structural analysis of narratives with the Coq proof assistant. In: van Eekelen M, Geuvers H, Schmaltz J, et al (eds) 2nd International Conference on Interactive Theorem Proving (ITP), Lecture Notes in Computer Science, vol 6898. Springer, pp 55–70, https://doi.org/10.1007/978-3-642-22863-6_7

Chaudhuri K, Lima L, Reis G (2017) Formalized meta-theory of sequent calculi for substructural logics. Electronic Notes in Theoretical Computer Science 332:57–73. https://doi.org/10.1016/j.entcs.2017.04.005

Chaudhuri K, Lima L, Reis G (2019) Formalized meta-theory of sequent calculi for linear logics. Theoretical Computer Science 781:24–38. https://doi.org/10.1016/J.TCS.2019.02.023

Cowley A, Taylor CJ (2011) Towards language-based verification of robot behaviors. In: International Conference on Intelligent Robots and Systems (IROS). IEEE Computer Society Press, pp 4776–4782, https://doi.org/10.1109/IROS.2011.6095028

Di Guardia R, Laurent O (2025) Type isomorphisms for multiplicative-additive linear logic. Logical Methods in Computer Science 21:24. https://doi.org/10.46298/lmcs-21(4:24)2025

Felty A, Olarte C, Xavier B (2021) A focused linear logical framework and its application to metatheory of object logics. Mathematical Structures in Computer Science 31(3):312–340. https://doi.org/10.1017/S0960129521000323

Fleury A, Retoré C (1994) The mix rule. Mathematical Structures in Computer Science 4(2):273–285. https://doi.org/10.1017/S0960129500000451

Girard JY (1987) Linear logic. Theoretical Computer Science 50:1–102. https://doi.org/10.1016/0304-3975(87)90045-4

Girard JY (1991) A new constructive logic: classical logic. Mathematical Structures in Computer Science 1(3):255–296. https://doi.org/10.1017/S0960129500001328

Girard JY (1996) Proof-nets: the parallel syntax for proof-theory. In: Ursini A, Agliano P (eds) Logic and Algebra, Lecture Notes In Pure and Applied Mathematics, vol 180. Marcel Dekker, New York, pp 97–124, https://doi.org/10.1201/9780203748671-4

Girard JY (1998) Light linear logic. Information and Computation 143(2):175–204. https://doi.org/10.1006/inco.1998.2700

Girard JY, Lafont Y (1987) Linear logic and lazy computation. In: Ehrig H, Kowalski R, Levi G, et al (eds) Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'87), Lecture Notes in Computer Science, vol 250. Springer, pp 52–66, https://doi.org/10.1007/BFb0014972

Kanovich M, Kuznetsov S, Nigam V, et al (2019) Subexponentials in non-commutative linear logic. Mathematical Structures in Computer Science 29(8):1217–1249. https://doi.org/10.1017/S0960129518000117

Lambek J (1958) The mathematics of sentence structure. The American Mathematical Monthly 65(3):154–170. https://doi.org/10.1080/00029890.1958.11989160

Lambek J, Scott P (1988) Introduction to Higher-Order Categorical Logic. No. 7 in Cambridge Studies in Advanced Mathematics, Cambridge University Press

Laurent O (2002) Étude de la polarisation en logique. Thèse de doctorat, Université Aix-Marseille II, URL https://theses.hal.science/tel-00007884v1

Laurent O (2004, revised 2017) A proof of the focusing property of linear logic, URL https://perso.ens-lyon.fr/olivier.laurent/llfoc2.pdf, unpublished note

Laurent O (2018) Around classical and intuitionistic linear logics. In: Proceedings of the thirty-third annual ACM/IEEE symposium on Logic In Computer Science. Association for Computing Machinery, https://doi.org/10.1145/3209108.3209132

Laurent O (2021) An anti-locally-nameless approach to formalizing quantifiers. In: Hritcu C, Popescu A (eds) CPP 2021: Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs. ACM, pp 300–312, https://doi.org/10.1145/3437992.3439926

Laurent O (2026) Multiplicative linear logic with self-dual unit, submitted

Lincoln P, Mitchell JC, Scedrov A, et al (1992) Decision problems for propositional linear logic. Annals of Pure and Applied Logic 56(1–3):239–311. https://doi.org/10.1016/0168-0072(92)90075-B

Melliès PA (2012) Game semantics in string diagrams. In: 27th Annual IEEE Symposium on Logic in Computer Science, pp 481–490, https://doi.org/10.1109/LICS.2012.58

Melliès PA, Tabareau N (2010) Resource modalities in tensor logic. Annals of Pure and Applied Logic 161(5):632–653. https://doi.org/10.1016/j.apal.2009.07.018

Nguyen LTD (2020) Unique perfect matchings, forbidden transitions and proof nets for linear logic with mix. Logical Methods in Computer Science 16(1). https://doi.org/10.23638/LMCS-16(1:27)2020

Pagani M (2007) Proofs, denotational semantics and observational equivalences in multiplicative linear logic. Mathematical Structures in Computer Science 17(2):341–359. https://doi.org/10.1017/S0960129506005652

Pollack R (1998) How to believe a machine-checked proof. In: Sambin G, Smith J (eds) Twenty Five Years of Constructive Type Theory. No. 36 in Oxford Logic Studies, Oxford University Press, https://doi.org/10.1093/oso/9780198501275.003.0013

Pous D (2012) Relation algebra for Coq. Coq library, URL http://perso.ens-lyon.fr/damien.pous/ra/

Power J, Webster C (1999) Working with linear logic in Coq. In: Theorem Proving in Higher Order Logics: Emerging Trends, URL http://www-sop.inria.fr/croap/TPHOLs99/proceeding.html

Schellinx H (1991) Some syntactical observations on linear logic. Journal of Logic and Computation 1(4):537–559. https://doi.org/10.1093/logcom/1.4.537

Xavier B, Olarte C, Reis G, et al (2017) Mechanizing focused linear logic in Coq. Electronic Notes in Theoretical Computer Science 338:219–236. https://doi.org/10.1016/J.ENTCS.2018.10.014

Yetter DN (1990) Quantales and (noncommutative) linear logic. Journal of Symbolic Logic 55(1):41–64. https://doi.org/10.2307/2274953