Mathematical Structures in Computer Science (2021), 1–29
doi:10.1017/S0960129521000323

CAMBRIDGE
UNIVERSITY PRESS

PAPER

A focused linear logical framework and its application to metatheory of object logics

Amy Felty[1], Carlos Olarte[2]* and Bruno Xavier[3]

[1]University of Ottawa, Ottawa, Canada, [2]LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, Villetaneuse, France and ECT, Universidade Federal do Rio Grande do Norte, Natal, Brazil, and [3]DIMAp, Universidade Federal do Rio Grande do Norte, Natal, Brazil
*Corresponding author. Email: olarte@lipn.univ-paris13.fr

(Received 21 February 2020; revised 6 September 2021; accepted 22 September 2021)

Abstract
Linear logic (LL) has been used as a foundation (and inspiration) for the development of programming languages, logical frameworks, and models for concurrency. LL's cut-elimination and the completeness of focusing are two of its fundamental properties that have been exploited in such applications. This paper formalizes the proof of cut-elimination for focused LL. For that, we propose a set of five cut-rules that allows us to prove cut-elimination directly on the focused system. We also encode the inference rules of other logics as LL theories and formalize the necessary conditions for those logics to have cut-elimination. We then obtain, for free, cut-elimination for first-order classical, intuitionistic, and variants of LL. We also use the LL metatheory to formalize the relative completeness of natural deduction and sequent calculus in first-order minimal logic. Hence, we propose a framework that can be used to formalize fundamental properties of logical systems specified as LL theories.

Keywords: Linear logic; cut-elimination; focusing; Coq

1. Introduction

Linear logic (LL) was proposed by Girard (1987) more than 30 years ago, but it still inspires computer scientists and proof theorists alike, being used as a foundation for programming languages, models of concurrency (Caires et al., 2016; Nigam et al., 2017; Olarte et al., 2018), and logical frameworks (Cervesato and Pfenning, 2002; Miller and Pimentel, 2013). LL is a resource conscious logic where formulas can be consumed during proofs. For instance, the linear implication $F \multimap G$ can be interpreted as the fact that *in order to produce G, F must be consumed*. The behavior of formulas in classical logic is recovered via the LL exponentials ! and ?: formulas marked with ? on the right side of the sequent can be weakened (discarded) or contracted (duplicated) at will. This allows us to faithfully encode in LL both intuitionistic and classical logics.

In order to be a meaningful tool for reasoning about systems, programming languages and other logics, two fundamental properties of LL are needed: cut-elimination Gentzen (1969) and the completeness of focusing. *Cut-elimination* states that any proof with instances of the cut-rule $\dfrac{\vdash \Gamma, F \quad \vdash \Delta, F^{\perp}}{\vdash \Gamma, \Delta}\ [Cut]$ can be transformed into a proof of the same formula without any instance of $[Cut]$. The two main consequences of this theorem are (1) the system's consistency, that is, it is not possible to prove both $\vdash F$ and $\vdash F^{\perp}$ and (2) all proofs satisfy the subformula property, that is, a proof of a formula F contains only sub-formulas of F. *Focusing*, on the other hand, is a proof search discipline proposed by Andreoli (1992) which constrains proofs by enforcing that rules sharing some structural property, like invertibility, are grouped together. The completeness of focusing states that if a formula has a proof, then it has a focused proof.

The combination of cut-elimination and focusing allows for the construction of powerful linear logical frameworks. By relying on these two properties, proof search is considerably improved. Moreover, these properties can be used to engineer the types of proofs, thus allowing the specification/encoding of a number of different proof systems (e.g., sequent calculus, natural deduction, and tableaux systems) for different logics (Miller and Pimentel, 2013; Nigam and Miller, 2010).

In previous work (Xavier et al., 2017), we formalized two sequent systems for first-order classical LL in Coq (Bertot and Castéran, 2004): the one-sided dyadic system that considers sequents of the form $\vdash \Theta : \Delta$ (separating the linear context Δ from the classical context Θ) and the triadic system also known as focused system. We proved cut-elimination for the dyadic system and completeness of focusing. We thus obtained the equivalence between the dyadic+cut, the dyadic, and the triadic systems. The novelties of that work included **(1) Quantifiers:** Most of the formalizations of cut-elimination procedures in the literature deal with propositional systems. The first-order quantifiers of LL were fundamental for encoding object logics (OLs) as LL theories. Quantifiers in that work were specified with the technique of Parametric HOAS (Chlipala, 2008) (e.g., dependent types in Coq). **(2) Completeness of focusing:** Cut-elimination theorems for a number of proof systems have been formalized, including propositional LL (Chaudhuri et al., 2019b). Our work was the first formalization of the LL's completeness of focusing. **(3) Encoding proof systems:** By relying on LL's focusing property, it is possible to adequately encode a number of proof systems (Miller and Pimentel, 2013; Nigam and Miller, 2010). We formalized such an adequacy theorem for intuitionistic propositional logic.

This paper is an extended version of Xavier et al. (2017). The main additional contributions are **(1) Syntax**: We use the Hybrid system for representing syntax of OLs (Felty and Momigliano, 2012). This approach does not use dependent types and has the advantage that it does not require axioms to state properties of closed terms and formulas, as is required when using Parametric HOAS. Hybrid's mechanisms for encoding syntax are implemented *definitionally* via a de Bruijn style encoding, and the required properties of the syntax are proved from these definitions. We show that Hybrid is general enough to define OLs featuring binders, including the first-order quantifiers and quantifiers on worlds in hybrid logics. In addition, Hybrid has so far mainly been used for reasoning about the metatheory of programming languages (see Mahmoud and Felty (2019); Felty and Momigliano (2009)). This is the first time we apply it to reasoning about logics, which requires new techniques for encoding OLs. **(2) Cut-elimination**: Instead of proving cut-elimination for the dyadic (unfocused system), we propose a set of five cut-rules dealing with focused and unfocused sequents. We then prove the cut-elimination theorem directly for the focused system. For this reason, in this paper, we do not specify the dyadic system nor the completeness of focusing. We proceed directly to the triadic system and show that the cut-rule is admissible in that system. **(3) Meta-level properties for OLs**: In Xavier et al. (2017), we showed how to specify propositional OLs as LL theories and prove adequacy of such specifications. Here, following Miller and Pimentel (2013), we give a step forward and formalize the proof of necessary conditions for cut-elimination of first-order OLs specified in LL. This way, we obtain cut-elimination theorems for some specific OL proof systems, including first-order classical (system LK), minimal (LM), and intuitionistic (LJ) logics, and classical and intuitionistic multiplicative-additive linear logic (MALL/iMALL). We also mechanize the expressiveness result in Chaudhuri et al. (2019a): hybrid linear logic (HyLL) is not more expressive than LL. Moreover, we propose an alternative encoding of HyLL which is cut-coherent and we prove HyLL's cut-elimination from it. Finally, we use the metatheory of LL to prove the mutual relative completeness of natural deduction and sequent calculus for first-order minimal logic. These applications are certainly a compelling example of the meta-level reasoning of our framework for drawing conclusions about fundamental properties of OLs.

Organization. Section 2 introduces the syntax and the use of Hybrid to formalize OLs and LL syntax. Section 3 formalizes the focused system and the proof of some structural properties. In addition to our new way of encoding syntax, the following two sections are completely new wrt Xavier et al. (2017). Section 4 proves the cut-elimination theorem for the focused system of LL. Section 5 formalizes the necessary conditions for cut-elimination of OLs and describes the aforementioned applications of our framework. Section 6 concludes the paper. Our formalization is available at https://github.com/meta-logic/coq-fll. We present here some of the most important definitions and key cases in proofs. For the sake of presentation, we omit some cases (e.g., in inductive definitions) and we also change marginally the notation to improve readability. Also, in Coq theorem statements, we assume that variables are implicitly universally quantified. The reader may always consult the complete definitions and proofs in the source files.

2. LL Syntax

LL Girard (1987) is a resource conscious logic, in the sense that formulas are consumed when used during proofs, unless they are marked with the exponential ? (whose dual is !). Formulas marked with ? behave *classically*, that is, they can be contracted (duplicated) and weakened (erased) during proofs. LL connectives include the additive conjunction & and disjunction $\oplus$ and their multiplicative versions $\otimes$ and $\parr$, together with their units and the first-order quantifiers:

$$
\begin{array}{rccccccccccccc}
F, G, \ldots \ ::= & A & \mid & F \otimes G & \mid & 1 & \mid & F \oplus G & \mid & 0 & \mid & \exists x.F & \mid & !F \\
\mid & A^{\perp} & \mid & F \parr G & \mid & \perp & \mid & F \& G & \mid & \top & \mid & \forall x.F & \mid & ?F \\
 & \text{LITERALS} & & \multicolumn{3}{c}{\text{MULTIPLICATIVES}} & & \multicolumn{3}{c}{\text{ADDITIVES}} & & \text{QUANTIFIERS} & & \text{EXPONENTIALS}
\end{array}
$$

Note that $(\cdot)^{\perp}$ (negation) has only atomic scope. For an arbitrary formula F, $F^{\perp}$ denotes the result of moving negation inward until it has atomic scope. The connectives in the first line denote the de Morgan dual of the connectives in the second line. Hence, for atoms A and B, the expression $(\perp \& (A \otimes (!B)))^{\perp}$ denotes $1 \oplus (A^{\perp} \parr (?B^{\perp}))$. The linear implication $F \multimap G$ is a short hand for $F^{\perp} \parr G$. The equivalence $F \equiv G$ is defined as $(F \multimap G) \& (G \multimap F)$.

The main issue when formalizing first-order logics in proof assistants is how to encode quantifiers. At first glance, one might consider the following naive signature for the constructors `fx` and `ex` of universally and existentially quantified formulas, respectively:

```
fx : (var → formula) → formula        ex : (var → formula) → formula
```

In order to define substitutions of variables for terms on such formulas, it is necessary to define a `term` type as the union of, for example, `vars` and `functions`, and also to implement substitution from scratch. This means dealing with variable capture and equality of terms.

It is possible to avoid this unnecessary bureaucracy if substitution is handled by the meta-level β-reduction. This means that a quantified formula $Qx.F$ ($Q \in \{\forall, \exists\}$) is represented as $Q(\lambda x.F)$, where λ is a meta-level binder. In this case, we have

```
fx : (term → formula) → formula      ex : (term → formula) → formula
```

This approach is called *higher-order abstract syntax* (HOAS) or λ-tree syntax (Miller and Palamidessi, 1999; Pfenning and Elliott, 1988). In a functional framework, the type `(term → formula)` ranges over all functions of this type. This is not desirable as it allows functions, called *exotic terms* (Despeyroux et al., 1995), to pattern-match on the input term and return a structurally (or logically) different formula for each case.

As a solution to this problem, we use the Hybrid system (Felty and Momigliano, 2012), implemented in Coq, to support reasoning about OLs expressed using HOAS. Hybrid is implemented as a two-level system, an approach first introduced in the $FO\lambda^{\Delta N}$ logic (McDowell and

Miller, 2002). Using this approach, the specification of the semantics of the OL and the meta-level reasoning about it are done within a single system but at different levels. In the case of Hybrid, an intermediate level is introduced by inductively defining a *specification logic* (SL) in Coq, and OL judgments are encoded in the SL. Hybrid has been used, for example, to prove properties of a quantum programming language with linear features (Mahmoud and Felty, 2019); in that case the SL was a simple LL. Here, the SL is our focused LL, and both the SL and the OLs we consider are more complex than in previous work.

Hybrid provides an underlying low-level de Bruijn representation of terms to which the HOAS representation of OL syntax can be mapped internally; the user works directly with the higher-level HOAS representation. Using such a representation, α-conversion at the meta-level directly represents bound variable renaming at the object level, and meta-level β-conversion can be used to directly implement object-level substitution. As a consequence, we avoid the need to develop large libraries of lemmas devoted to operations dealing with variables, such as capture-avoiding substitution, renaming, and fresh name generation. The de Bruijn representation is introduced in the file `Hybrid.v` as the type `expr`, defined inductively in Coq as follows:

```
Inductive expr : Set :=
  | CON : con → expr    (* Constants *)         | APP : expr → expr → expr  (* Application *)
  | VAR : var → expr    (* Free variables *)    | ABS : expr → expr.        (* Abstraction *)
  | BND : bnd → expr    (* Bound variables *)
```

Here, `BND` and `VAR` represent bound and free variables, respectively, and `bnd` and `var` are defined to be the natural numbers. Thus, we represent infinite sets of variables using numbers and not explicit variable names. The type `con` is a parameter to be filled in when defining the constants used to represent an OL. The library then includes a series of definitions used to define the operator `lambda` of type `(expr → expr) → expr`, which provides the capability to express OL syntax using HOAS, including negative occurrences in the types of binders. Expanding its definition fully down to primitives gives the low-level de Bruijn representation, which is hidden from the user when reasoning about metatheory. In fact, the user only needs `CON`, `VAR`, `APP`, and `lambda` to define the OL syntax. Two other predicates from Hybrid that will appear in our developments are `proper: expr → Prop` and `uniform: (expr → expr) → Prop`. The `proper` predicate rules out terms that have occurrences of bound variables that do not have a corresponding binder (*dangling indices*). The `uniform` predicate is applied to arguments of `lambda` and rules out exotic terms, in this case functions of type `(expr → expr)` that do not encode object-level syntax (i.e., functions that are not *uniform* on their argument x, returning structurally different terms depending on x).

As mentioned, the type `con` is actually a parameter in the Hybrid library. In particular, it first appears inside a section in `Hybrid.v`, and as a result, outside this section, `expr` has type `Set → Set`. Once `con` is defined, then `(expr con)` is the actual type (element of `Set`) used to express OL terms. In our case, the constants of the OLs we consider will be introduced later as an inductive type called `Econ` and thus `(expr Econ)` is the type of terms and formulas of this OL. The type `con` is also a parameter to any definition that uses it.

We define our focused LL as an SL in the next section. Here, we present the encoding of its syntax and related definitions (file `Syntax.v`). LL formulas are defined by the inductive type `oo`:

```
Inductive oo : Set :=
| atom : atm → oo        | Bot :  oo                 | Bang :  oo → oo
| perp : atm → oo        | AAnd : oo → oo → oo       | Quest : oo → oo
| Top :  oo              | MAnd : oo → oo → oo       | All :   (expr con → oo) → oo
| One :  oo              | AOr :  oo → oo → oo       | Some :  (expr con → oo) → oo.
| Zero : oo              | MOr :  oo → oo → oo
```

The type `atm` is defined for each OL and typically includes the atomic propositions of the OL. Most of the constructors are straightforward encodings of the units and connectives of LL discussed above. The `All` and `Some` constructors each take a function as an argument, and thus the bound variable in the quantified formula is encoded using lambda abstractions in Coq.

Just as we do at the object level, we must also rule out exotic terms at the SL level. Unlike at the OL level where it is defined on functions involving the de Bruijn representation, here we define it for type `oo` as the following `uniform_oo` predicate. We only show some cases:

```
Inductive uniform_oo: (expr con → oo) → Prop :=
| uniform_atom: forall (a: expr con → atm), uniform_atm a → uniform_oo (fun x ⇒ (atom (a x)))
| uniform_Top: uniform_oo (fun x ⇒ Top)
| uniform_AAnd: forall A B, uniform_oo A → uniform_oo B → uniform_oo (fun x ⇒ (AAnd (A x) (B x)))
| uniform_All: forall (A: expr con → expr con → oo),
    (forall y:expr con, uniform_oo (fun x ⇒ (A y x))) → uniform_oo (fun x ⇒ (All (fun y ⇒ (A y x)))) [...]
```

Like `atm`, the `uniform_atm` predicate is a parameter to the SL; it must be defined for each OL. It typically requires that Hybrid's `uniform` predicate holds on all subterms of type `expr con → expr con`. The other cases reflect the fact that functions defining quantifiers cannot do pattern matching, thus returning the same shape of formula (regardless of the actual parameter) and replacing the bound variable only at the atomic level.

Another consequence of the functional representation of formulas is that Coq's axiom of Functional Extensionality is needed to check whether two formulas are the same:

```
Axiom functional_extensionality_dep : forall {A} {B : A → Type},
  forall (f g : forall x : A, B x), (forall x, f x = g x) → f = g.
```

Given two function f and g, we conclude $f = g$ whenever $f(x) = g(x)$ for all x. In our setting, the types of f and g are `expr con → oo`. We also note that the version of Hybrid used here requires a description axiom from Coq's classical library (Felty and Momigliano, 2012). This axiom is used for historical reasons, because this version of Hybrid was adapted directly from a version written in HOL, a proof assistant that implements a classical higher-order logic.

The rest of the file `Syntax.v` contains some definitions useful for the forthcoming results. In particular, we define both the complexity and the De Morgan dualilties of LL formulas:

```
Fixpoint complexity (X:oo) :=
    match X with
    | atom A    ⇒ 1
    | MAnd F G ⇒ 1 + complexity F + complexity G
    | Some FX   ⇒ 1 + complexity (FX (VAR con 0)) [...]
```

```
Fixpoint dual (X: oo) :=
   match X with
   | perp A     ⇒ atom A
   | MOr F G    ⇒ MAnd (dual F) (dual G)
   | All FX     ⇒ Some (fun x ⇒ dual (FX x)) [...]
```

Note that in `complexity`, we apply the function `FX` to the free variable 0 (`VAR con 0`). In lemma `ComplexityUniformEq` we show that, given an `uniform_oo` function `FX` and two `proper` terms X and Y, `complexity (FX x) = complexity (FX y)`. Some other definitions and lemmas in this file will be introduced when needed in the following sections.

As mentioned in the introduction, in our previous work, we used Parametric HOAS (Chlipala, 2008) to encode the syntax of LL. There, the types `term` (for pre-terms) and `llexp` (for pre-formulas) are inductively defined. The constructor `var (t:T)` in `term` denotes a place holder, and the LL quantifiers in `llexp` have type `(T → llexp) → llexp`. In these definitions, `T:Type` is a parameter of the specification. Since the representation of formulas must be independent of `T`, the actual type for LL formulas is the dependent type `LLExp := forall T:Type, llexp T` (similarly for terms). The dependent product, closing `T`, guarantees that functions in quantifiers cannot do pattern matching on their parameters, thus avoiding exotic terms. In this setting, it is not possible to reason inductively on `llexp` expressions. For this reason, the predicate `Closed: LLExp → Prop` is inductively defined asserting that formulas cannot contain free variables (`var x`), since such terms can appear only in the scope of quantifiers. This fact cannot be proved and it has to be assumed as an axiom. Proofs then proceed by induction on the fact that the given LL formula is *closed*. This machinery allows

$$\frac{\vdash \Gamma_1, F \quad \vdash \Gamma_2, G}{\vdash \Gamma_1, \Gamma_2, F \otimes G}\ [\otimes] \qquad \frac{\vdash \Gamma, F, G}{\vdash \Gamma, F \parr G}\ [\parr] \qquad \frac{\vdash \Gamma, F \quad \vdash \Gamma, G}{\vdash \Gamma, F \,\&\, G}\ [\&] \qquad \frac{\vdash \Gamma, F}{\vdash \Gamma, F \oplus G}\ [\oplus_1] \qquad \frac{\vdash \Gamma, G}{\vdash \Gamma, F \oplus G}\ [\oplus_2]$$

$$\frac{\vdash ?F_1, \ldots, ?F_n, F}{\vdash ?F_1, \ldots, ?F_n, !\, F}\ [!] \qquad \frac{\vdash \Gamma, F}{\vdash \Gamma, ?F}\ [?] \qquad \frac{\vdash \Gamma, F[c/x]}{\vdash \Gamma, \forall x.F}\ [\forall] \qquad \frac{\vdash \Gamma, F[x/t]}{\vdash \Gamma, \exists x.F}\ [\exists]$$

$$\frac{}{\vdash A^{\perp}, A}\ [I] \qquad \frac{}{\vdash \Gamma, \top}\ [\top] \qquad \frac{}{\vdash 1}\ [1] \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \perp}\ [\perp] \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?F}\ [W] \qquad \frac{\vdash \Gamma, ?F, ?F}{\vdash \Gamma, ?F}\ [C]$$

Figure 1. Linear logic sequent calculus: c in $[\forall]$ is fresh, that is, does not appear in Γ.

for building the syntax of LL, but it does not offer any support for defining the syntax of the OL that, in turns, is specified as an inductive set. Therefore, the problems related to binders appear again at the OL level and we dealt only with propositional OLs in Xavier et al. (2017). As we shall see, with Hybrid, we can encode first-order OLs straightforwardly.

3. Sequent Calculi

The proof system for one-sided (classical) first-order LL is depicted in Figure 1. A sequent has the form $\vdash \Gamma$ where Γ is a multiset of formulas (i.e., exchange is implicit). While this system is the one normally used in the literature, LL's focused proof system is equipped with some more structure. As shown by Andreoli (1992), it is possible to incorporate the structural rules of contraction $[C]$ and weakening $[W]$ into the introduction rules. The key observation is that formulas of the form $?F$ can be contracted and weakened. This means that such formulas can be treated as in classical logic, while the remaining formulas are treated linearly. This is reflected in the syntax of the so-called *dyadic sequents* which have two contexts, namely, the classical and the linear context. Consider for instance the following (dyadic) rules:

$$\frac{}{\vdash \Theta : A^{\perp}, A}\ [I] \qquad \frac{\vdash \Theta, F : \Gamma}{\vdash \Theta : \Gamma, ?F}\ [?] \qquad \frac{\vdash \Theta, F : \Gamma, F}{\vdash \Theta, F : \Gamma}\ [copy] \qquad \frac{\vdash \Theta : \Gamma_1, A \quad \vdash \Theta : \Gamma_2, B}{\vdash \Theta : \Gamma_1, \Gamma_2, A \otimes B}\ [\otimes]$$

Here, Θ is a set of formulas and Γ a multiset of formulas. The sequent $\vdash \Theta : \Gamma$ is interpreted as the LL sequent $\vdash ?\Theta, \Gamma$ where $?\Theta = \{?F \mid F \in \Theta\}$. It is then possible to define a proof system for LL without explicit weakening (implicit in rule $[I]$ above) and contraction (implicit in $[copy]$ and $[\otimes]$). Notice that only the linear context Γ is split among the premises in $[\otimes]$. The complete proof system can be found in Andreoli (1992), and its formalization in Coq is described in Xavier et al. (2017). We will not elaborate more about this system, since we shall work directly on the focused (triadic) system described below.

3.1. Focused system

Focusing is a discipline on proofs aiming at reducing non-determinism during proof search and it allows specifiers to engineer proofs, as we illustrate in Section 5. Proofs are organized in two alternating phases: the negative phase contains only invertible rules and the positive phase contains only non-invertible rules. The connectives $\parr$, $\perp$, &, $\top$, ?, and $\forall$ have invertible introduction rules and are thus classified as *negative*. The remaining connectives $\otimes$, 1, $\oplus$, !, and $\exists$ are *positive*. Formulas inherit their polarity according to their main connective, for example, $F \otimes G$ is positive and $F \parr G$ is negative. Although the bias assigned to atoms does not interfere with provability (Miller and Saurin, 2007), also formalized in Xavier et al. (2017), here we follow Andreoli's convention of classifying atomic formulas as negative and then, negated atoms as positive.

In LL's focused proof system FLL (also called triadic system), there are two types of sequents where Θ is a set of formulas, Γ a multiset of formulas, and L a list of formulas:

$$\frac{}{\vdash \Theta : \Gamma \Uparrow \top, L}\;[\top] \qquad \frac{\vdash \Theta : \Gamma \Uparrow F, G, L}{\vdash \Theta : \Gamma \Uparrow F \parr G, L}\;[\parr] \qquad \frac{\vdash \Theta, F : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow ?F, L}\;[?] \qquad \frac{\vdash \Theta : \Gamma \Uparrow F, L \quad \vdash \Theta : \Gamma \Uparrow G, L}{\vdash \Theta : \Gamma \Uparrow F \,\&\, G, L}\;[\&]$$

$$\frac{\vdash \Theta : \Gamma \Uparrow F[c/x], L}{\vdash \Theta : \Gamma \Uparrow \forall x\, F, L}\;[\forall] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow \bot, L}\;[\bot] \qquad \frac{}{\vdash \Theta : \cdot \Downarrow 1}\;[1] \qquad \frac{\vdash \Theta : \Gamma \Downarrow F \quad \vdash \Theta : \Gamma' \Downarrow G}{\vdash \Theta : \Gamma, \Gamma' \Downarrow F \otimes G}\;[\otimes] \qquad \frac{\vdash \Theta : \cdot \Uparrow F}{\vdash \Theta : \cdot \Downarrow !\, F}\;[!]$$

$$\frac{\vdash \Theta : \Gamma \Downarrow F_i}{\vdash \Theta : \Gamma \Downarrow F_1 \oplus F_2}\;[\oplus_i] \qquad \frac{\vdash \Theta : \Gamma \Downarrow F[t/x]}{\vdash \Theta : \Gamma \Downarrow \exists x\, F}\;[\exists] \qquad \frac{}{\vdash \Theta : A \Downarrow A^{\perp}}\;[I_1] \qquad \frac{}{\vdash \Theta, A : \cdot \Downarrow A^{\perp}}\;[I_2]$$

$$\frac{\vdash \Theta : S, \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow S, L}\;[R \Uparrow] \qquad \frac{\vdash \Theta : \Gamma \Downarrow P}{\vdash \Theta : \Gamma, P \Uparrow}\;[D_1] \qquad \frac{\vdash \Theta, P : \Gamma \Downarrow P}{\vdash \Theta, P : \Gamma \Uparrow}\;[D_2] \qquad \frac{\vdash \Theta : \Gamma \Uparrow N}{\vdash \Theta : \Gamma \Downarrow N}\;[R \Downarrow]$$

Figure 2. The focused proof system **FLL**. *A* is an atom (with negative polarity), *N* is a negative formula, *P* is a positive formula or a negated atom ($A^{\perp}$), and *S* in rule $[R \Uparrow]$ is a positive formula or a literal (*A* or $A^{\perp}$).

- $\vdash \Theta : \Gamma \Uparrow L$ belongs to the *negative phase*. During this phase, all negative formulas in L are introduced and all positive formulas and literals are moved to Γ.
- $\vdash \Theta : \Gamma \Downarrow F$ belongs to the *positive phase*. During this phase, all positive connectives at the root of F are introduced.

The system is in Figure 2. Let us explain some of the rules from the conclusion to the premises. The rules belonging to the negative phase ($\vdash \Theta : \Gamma \Uparrow L$) pick the first formula F of the list L. If F is a negative formula, then it is decomposed (see, e.g., the rules $[\bot]$, $[\parr]$, $[\forall]$). Note also that the rule $[?]$ stores the formula F into the classical context. The case when F is a literal or a positive formula is handled by the rule $[R \Uparrow]$ that stores this formula into the linear context.

The negative phase ends when the list L is empty. In that case, the proof moves to the positive phase by using the rules $[D_1]$ and $[D_2]$. In both cases, a positive formula or a negated atom P is selected (focused on). In $[D_1]$, P is taken from the linear context (and thus erased from it). In $[D_2]$, a copy of P is taken from the classical context (thus making an implicit contraction). In all these rules, the focus persists on the decomposed subformulas (e.g., F and G in the rule $[\otimes]$).

The positive phase finishes by using the rule $[1]$ or the initial rules ($[I_1]$ and $[I_2]$). It is also possible that the focused formula belongs to the negative phase. In that case, the rule $[R \Downarrow]$ (also called *release*) is used to switch the polarity of the proof.

For an atom A, the sequent $\vdash \Gamma : \Delta \Downarrow A^{\perp}$ must finish immediately by either $[I_1]$ ($\Delta = A$) or $[I_2]$ ($A \in \Gamma$ and $\Delta = \cdot$). On the other hand, the sequent $\vdash \Gamma : \Delta \Downarrow A$ loses focusing and produces, as unique premise, the sequent $\vdash \Gamma : \Delta, A \Uparrow$ after an application of $[R \Downarrow]$ followed by $[R \Uparrow]$. Due to our choice of polarity for atoms, note that the rule $[D_1]$ cannot be used to focus on A in $\vdash \Gamma : \Delta, A \Uparrow$ (similarly for $[D_2]$). Differently, in the sequent $\vdash \Gamma : \Delta, A^{\perp} \Uparrow$, the negated atom $A^{\perp}$ can be chosen for focusing and, in that case, the sequent must finish with the initial rules.

3.2. FLL system in Coq

The sequent rules of the triadic system (see file `Sequent.v`) are specified as follows:

```
Section SequentSystem.
  Variable th : oo → Prop . (* the theory *)
  Inductive seq: multiset oo → multiset oo → Arrow → Prop := [...] (* see below *)
End  SequentSystem.
```

The *theory* `th: oo` → `Prop` defines a set of LL formulas that represent axioms of an OL up to some side condition that usually expresses well-formedness for a term of the OL. The proof of the proposition `(th F)` cannot be discharged by using the logical rules of LL. Instead, it is proved by using the machinery of Hybrid and the inductive definition of well-formedness for the OL (some examples in Section 5). We note that in previous applications of Hybrid to reasoning about OLs, for example, Felty and Momigliano (2012), th is called prog because it is expressed in the form of a simple logic program that encodes the inference rules of the semantics of a programming language. Since some of the OLs in Section 5 are logics, we require a more flexible approach that allows us to encode OL's inference rules and their side conditions as shown below.

Due to Coq's section mechanism, the variable th is nothing more than a parameter, and thus the type of seq is (oo → Prop) → multiset oo → multiset oo → Arrow → Prop, indicating that the first argument of seq is the underlying theory, the second and third arguments (lists up to permutations) correspond to the classical and linear contexts, respectively, and the last argument is a term of type Arrow. The latter type is inductively defined in Syntax.v with two constructors: UP L (representing the negative phase $\Uparrow L$) and DW F (for the positive phase $\Downarrow F$).

Let us show some examples of the specification of the rules starting with the initial rules:

```
| tri_init1' : forall G A,  seq G [atom A] (DW (perp A))
| tri_init2' : forall G A,  In (atom A) G → seq G [ ] (DW (perp A))
```

Here, A:atm is an atomic proposition (defined in the OL) and perp A stands for $A^{\perp}$. In x L is the Coq predicate stating that x belongs to the list L. Below are some of the rules of the positive phase:

```
| tri_tensor' : forall B M N MN F G, Permutation MN (M + + N) →
                    seq B M (DW F) →   seq B N (DW G) → (* two premises *)
                    seq B MN (DW (MAnd F G))          (* conclusion *)
| tri_rel' : forall B F L, release F → seq B L (UP [F])  → seq B L (DW F)
| tri_ex'  :  forall B FX M t, uniform_oo FX → proper t → seq B M (DW (FX t)) → seq B M (DW (Some  FX))
```

The rule for $[\otimes]$ (tri_tensor') embeds the exchange rule by decreeing that the linear context MN in the conclusion must be a permutation of the concatenation (++) of the contexts in the two premises. The release rule $[R\Downarrow]$ checks if F must lose focusing (predicate release) and, bottom-up, the proof continues with a sequent belonging to the negative phase. In the case of the existential quantifier, we check that FX satisfies the uniform_oo condition and also, that the term t is proper. In that case, the proof continues focused on the formula FX t (corresponding to $F[t/x]$).

For the negative phase, let us present the following rules:

```
| tri_par' : forall B L M F G, seq B L (UP (F:: G:: M))  → seq BL (UP ((MOr F G) ::  M))
| tri_store' : forall B L M F,  ~ asynchronous F→  seq B (L ++ [F]) (UP M)  → seq B L (UP (F:: M))
```

As expected, the $[⅋]$ (tri_par') rule decomposes $F ⅋ G$ into F and G. The store rule $[\Uparrow R]$ stores the positive formula F (i.e., F is not asynchronous/negative) into the linear context.

Decision rules. The FLL system has three different decision rules:

```
| tri_dec1' : forall B L L' F, ~IsPositiveAtom F → remove F L L' → seq B L' (DW F)  → seq B L (UP [   ])
| tri_dec2' : forall B L F, ~IsPositiveAtom F → In F B → seq B L (DW F)  → seq B L (UP [  ])
| tri_dec3' : forall B L F, th F  → ~IsPositiveAtom F → seq B L (DW F) → seq B L (UP [ ])
```

In the first case (corresponding to $[D_1]$), the inductively defined predicate remove F L L' states that L' results from L after removing F. Condition ~IsPositiveAtomF guarantees that F is not a literal of the form atom A. The rule tri_dec2 specifies the decision rule $[D_2]$ for the classical context. Finally, the rule tri_dec3 (that we shall also denote as $[D_3]$) checks whether F belongs to the theory (th F) in order to, bottom-up, continue with the proof of $\vdash B : L \Downarrow F$.

Inductive Measures. In our proofs, we usually proceed by induction on the height of the derivations. Hence, we also specify an alternative system where this measure is explicit:

```
Inductive seqN: nat → multiset oo → multiset oo → Arrow → Prop :=
  | tri_init1 : forall B A n,  seqN n B [atom A] (DW (perp A))
  | tri_tensor : forall B M N MN F G n,  Permutation MN (M+ +N) →
        seqN n B M (DW F) → seqN n B N (DW G) → seqN (S n) B MN (DW (MAnd F G))
  [...]
```

Note that the initial rule ends with any height n (and not necessarily with $n = 0$) and in rules with two premises, both premises have the same height. This is without loss of generality since we can show the following: if a sequent is provable with height n, then it is provable with height $m \geq n$:

```
Theorem HeightGeq : seqN th n Gamma Delta arrow → forall m, m≥n → seqN th m Gamma Delta arrow.
```

We also include the following properties, relating the two definitions of sequents.

```
Theorem seqNtoSeq : seqN th n Gamma Delta arrow → seq prog Gamma Delta arrow.
Theorem seqtoSeqN : seq th Gamma Delta arrow → exists n, seqN prog n Gamma Delta arrow.
```

The first has a simple proof by induction on n. In the second, we proceed by induction on the evidence that the sequent $\vdash \Gamma : \Delta \ \Updownarrow$ (where $\Updownarrow$ can be $\Uparrow L$ or $\Downarrow F$) is provable. All the cases are easy, but the case of `All` is representative of a class of admissible theorems such that attempting their proofs in Coq results in subgoals of the following form:

```
From the hypotheses: FX:expr con → oo,   x, y: expr con,   Hx:proper x ,   Hy:proper y,   H:P (FX x)
Prove: P (FX y)
```

Here `P (FX x)` expresses some property of `(FX x)`, and `x` and `y` are two variables in the context for which we have no information except that they both satisfy the `proper` predicate. Furthermore, the context is such that we can prove on paper (but not in Coq) that `forall x, proper x → P (FX x)`, from which the conclusion follows. Thus, the proof of `seqtoSeqN` relies on an axiom expressing an admissible theorem. We note that the analogous version of this theorem in the Parametric HOAS approach (Xavier et al., 2017) also requires an axiom.

Notation 1. We shall use `|-- B ; D ; (> L)` and `|-- B ; D ; (>> L)` to denote, respectively, the sequents $\vdash B : D \Uparrow L$ and $\vdash B : D \Downarrow F$. Also, we shall use `N |-- ...` (resp. `th |-- ...`) when the height n of the derivation (resp. theory `th`) needs to be explicit. `[ ]` denotes the empty list/context. The LL formulas $\forall x.FX$, $\exists x.FX$, $F \otimes G$ and $F \parr G$ will be written as `F{FX}`, `E{FX}`, `F ** G` and `F|G`, respectively.

Tactics and automation. We have included in file `FLLTactics.v` several tactics that simplify proofs involving FLL sequents. Let us explain the most relevant ones.

The tactic `solveF` solves most of the auxiliary subgoals generated during proofs in FLL. For instance, consider the rule $[R \Uparrow]$ in Figure 2 where the formula S must be a positive formula or a literal. Hence, the application of this rule generates a subgoal of the form `asynchronous S` checking that S can be *stored* into the linear context. If S is indeed a formula that does not belong to the negative phase, this subgoal can be easily solved:

```
Ltac solveF :=
  repeat match goal with
    | [ |−~ asynchronous _] ⇒ intro H; inversion H;auto
  [...]
```

Using Coq's Ltac language[1] for defining tactics, `solveF` performs a pattern matching on the current goal being proved. If it is of the form `~asynchronous S`, then the sequence of tactics `intro H; inversion H;auto` is applied. Other subgoals considered in `solveF` include, for example, checking whether a formula is not a positive atom (see the decision rules), checking whether the predicate `Permutation` holds between two list of formulas (see rule $[\otimes]$) and solving arithmetic goals:

```
| [ |-  IsPositiveAtom _ ] ⇒ intro H; inversion H;auto
| [ |- Permutation _ _] ⇒ perm
| [ |- _ >= _] ⇒ subst; lia
```

In the case of permutations, the tactic `perm` adapted from the CertiCoq project[2] is applied; for arithmetic goals, the Coq's tactic `lia` for linear integer arithmetic is applied.

`solveF` also solves some (trivial) goals that assume an inconsistent set of hypotheses. This is the case, for instance, of hypotheses of the form `In x []` (asserting that x belongs to the empty list), `a :: L = []` (asserting that a non-empty list is equal to an empty list), etc. Such contradictions can be easily detected and used to finish the current goal:

```
| [H :  In _ [] |- _ ] ⇒ inversion H   |     [H :  _ ::  _ = [] |- _] ⇒ inversion H
```

The goal of `solveF` is to avoid all the unnecessary bureaucracy during proofs and allow users to apply directly FLL rules as in paper proofs. For that, suitable tactical notations are provided, wrapping the use of `solveF`:

```
Tactic Notation "store" := apply tri_store ;solveF.
Tactic Notation "decide1" constr(G) := eapply tri_dec1 with (F:= G);solveF.
```

By typing `store`, Coq will apply the rule $[R \Uparrow]$ and, if possible, prove that the current formula can be indeed stored. As another example, on a goal of the form $\vdash \Theta : \Delta \Uparrow$, the tactic `decide1 F` tries to apply $[D_1]$ and checks whether F is not a positive atom and $F \in \Delta$.

The tactic `solveLL` leverages the use of `solveF` to automate the proof of FLL sequents:

```
Ltac solveLL :=
  try match goal with
      | [|-  seqN _ _ _ _ (>> (perp _))]  ⇒ first [apply tri_init1;auto | apply tri_init2;auto]
      | [|-  seqN _ _ _ [perp ?A]  (>>  (atom  ?A))] ⇒ apply InitPosNegDwN
      | [|-  seqN _ _ _ [atom ?A  ;  perp  ?A] (> [])] ⇒ apply InitPosNegN
   [...]
```

In the first case, on a sequent of the form $\vdash \Theta : \Delta \Downarrow A^{\perp}$, `solveLL` tries to apply the initial rules of the system ($[I_1]$ and $[I_2]$). In the second and the third case, the following lemmas (proved in `StructuralRules.v`) are considered:

```
Theorem InitPosNegDwN : seqN theory 4 Gamma  [perp A ]  (DW (atom A)).
Theorem InitPosNegN : seqN theory 2 Gamma [atom A ; perp A ] (UP []).
```

`InitPosNegDwN` shows that the sequent $\vdash \Gamma : A^{\perp} \Downarrow A$ is provable in at least four steps using the rules $[R \Downarrow]$, $[R \Uparrow]$, $[D_1]$, and $[I_1]$. Similarly for the theorem `InitPosNegN`.

On a focused sequent of the form $\vdash \Theta : \Delta \Downarrow F$, if F belongs to the negative phase, focusing must be lost. This is the purpose of the following subcases of `solveLL`:

```
| [|-  seqN _ _ _ _ (>>  ?F)]  ⇒ match F with  | MOr _ _ ⇒ release;solveLL
                                              | All _ ⇒ release;solveLL
                                              [...]
```

where the release rule $[R \Downarrow]$ is applied and `solveLL` is recursively called to perform the subsequent steps of the negative phase.

Since the negative phase of the proof does not require any interaction with the user, much of this phase can be automatized by applying the respective rule:

```
| [ |- seqN _ _ _ _ (> (( atom _ )  ::  _ ))]  ⇒ store ;solveLL
| [ |- seqN _ _ _ _ (> (( perp _ )  ::  _ ))]  ⇒ store ;solveLL
| [ |- seqN _ _ _ _ (> (  (Quest _) ::  _))  ] ⇒ apply tri_quest ;solveLL
| [ |- seqN _ _ _ _ (> (  (Top) ::   _))    ] ⇒ apply tri_top
```

In the first two cases, literals are stored (rule $[R\Uparrow]$); in the third, formulas marked with ? are stored into the classical context ([?]); if the current formula is $\top$ the proof ends with an application of $[\top]$, etc.

Let us give an example to illustrate both proofs in FLL and the use of the tactics implemented. Consider the following simple OL defining the natural numbers:

```
Inductive Atom : Set := p : uexp → Atom. (* unary predicate symbol p *)
Inductive Econ : Set :=  | z : Econ   | s : Econ. (* Constants used to build OL terms *)
(* Actual operators of the OL: Zero and Successor *)
Definition Z  := (CON z) .
Definition S : (expr Econ) → (expr Econ) :=  fun N:( expr Econ) ⇒ (APP (CON s) N) .
Definition step      := fun t:uexp ⇒ perp (p t) ** atom (p (S t)). (* i.e., (p(t)⊥ ⊸ p(s(t))⊥)⊥ *)
Definition stepPerp := fun t:uexp ⇒ atom (p t) ** perp (p (S t)). (* i.e., (p(t) ⊸ p(s(t)))⊥ *)
```

Consider the proof of the sequent $\vdash \cdot : \cdot \Uparrow ?\exists FX, p(0)^{\perp} \multimap p(2)^{\perp}$ where $\exists FX$ is a shorthand for $\exists t.(p(t)^{\perp} \otimes p(S(t)))$ (see `step` above) and 0, 1, and 2 represent Z, $S\,Z$, and $S\,(S\,Z)$, respectively:

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{$[I_1]$}
\UnaryInfC{$\vdash \exists FX : p(0) \Downarrow p(0)^{\perp}$}
\AxiomC{}
\RightLabel{$[I_1]$}
\UnaryInfC{$\vdash \exists FX : p(1) \Downarrow p(1)^{\perp}$}
\AxiomC{}
\doubleLine
\RightLabel{$[D_1, I_1]$}
\UnaryInfC{$\vdash \exists FX : p(2)^{\perp}, p(2) \Uparrow \cdot$}
\doubleLine
\RightLabel{$[R\Downarrow, R\Uparrow]$}
\UnaryInfC{$\vdash \exists FX : p(2)^{\perp} \Downarrow p(2)$}
\doubleLine
\RightLabel{$[\exists(t := 1), \otimes]$}
\BinaryInfC{$\vdash \exists FX : p(2)^{\perp}, p(1) \Downarrow \exists FX$}
\RightLabel{$[D_2]$}
\UnaryInfC{$\vdash \exists FX : p(2)^{\perp}, p(1) \Uparrow \cdot$}
\doubleLine
\RightLabel{$[R\Downarrow, R\Uparrow]$}
\UnaryInfC{$\vdash \exists FX : p(2)^{\perp} \Downarrow p(1)$}
\RightLabel{$[\otimes]$}
\BinaryInfC{$\vdash \exists FX : p(0), p(2)^{\perp} \Downarrow p(0)^{\perp} \otimes p(1)$}
\RightLabel{$[\exists(t := 0)]$}
\UnaryInfC{$\vdash \exists FX : p(0), p(2)^{\perp} \Downarrow \exists FX$}
\RightLabel{$[D_2]$}
\UnaryInfC{$\vdash \exists FX : p(0), p(2)^{\perp} \Uparrow \cdot$}
\doubleLine
\RightLabel{$[?, \parr, 2 \times R\Uparrow]$}
\UnaryInfC{$\vdash \cdot : \cdot \Uparrow ?\exists FX, p(0)^{\perp} \multimap p(2)^{\perp}$}
\end{prooftree}
$$

```
Example seq1 : |-- [];  [];   > [? E{ step};
     perp (p Z) -o perp (p ( S ( S Z)))].
Proof with solveLL.
 simpl... (* unfolding -o + solveLL*)
 decide2 (E{ step}).
 existential Z .
 tensor [atom (p Z)] [perp (p(S ( S Z)))].
 decide2 (E{ step}) .
 existential (S Z).
 tensor [atom (p(S Z))] [perp (p(S(S Z)))].
Qed.
```

Note the forward chaining proof style where, from $p(0)$, we conclude $p(1)$ to later show $p(2)$. In both steps, the formula $\exists FX$ is focused on. The proof in Coq only requires some information during the positive phase: which formula must be focused on; the witness for the rule $[\exists]$; and how to split the context in $[\otimes]$. The other intermediary goals are solved automatically. In the first line, the Coq's notation "…" applies the tactic specified in the header of the proof (`Proof with solveLL`). Note also that the goal $\vdash \exists FX : p(2)^{\perp} \Downarrow p(2)$ is automatically discharged.

Consider the proof below where $\exists FX'$ stands for $\exists t.(p(t) \otimes p(S(t))^{\perp})$ (see `stepPerp` above):

$$
\begin{prooftree}
\AxiomC{}
\doubleLine
\RightLabel{$[R\Downarrow, R\Uparrow, D_1, I_1]$}
\UnaryInfC{$\vdash \exists FX' : p(0)^{\perp} \Downarrow p(0)$}
\AxiomC{}
\RightLabel{$[I_1]$}
\UnaryInfC{$\vdash \exists FX' : p(1) \Downarrow p(1)^{\perp}$}
\RightLabel{$[\otimes]$}
\BinaryInfC{$\vdash \exists FX' : p(0)^{\perp}, p(1) \Downarrow p(0) \otimes p(1)^{\perp}$}
\doubleLine
\RightLabel{$[D_2, \exists(t := 0)]$}
\UnaryInfC{$\vdash \exists t.(...) : p(0)^{\perp}, p(1) \Uparrow \cdot$}
\doubleLine
\RightLabel{$[R\Downarrow, R\Uparrow]$}
\UnaryInfC{$\vdash \exists FX' : p(0)^{\perp} \Downarrow p(1)$}
\AxiomC{}
\RightLabel{$[I_1]$}
\UnaryInfC{$\vdash \exists FX' : p(2) \Downarrow p(2)^{\perp}$}
\RightLabel{$[\otimes]$}
\BinaryInfC{$\vdash \exists FX' : p(0)^{\perp}, p(2) \Downarrow p(1) \otimes p(2)^{\perp}$}
\doubleLine
\RightLabel{$[D_2, \exists(t := 1)]$}
\UnaryInfC{$\vdash \exists FX' : p(0)^{\perp}, p(2) \Uparrow \cdot$}
\doubleLine
\RightLabel{$[?, \parr, 2 \times R\Uparrow]$}
\UnaryInfC{$\vdash \cdot : \cdot \Uparrow ?\exists FX', p(0) \multimap p(2)$}
\end{prooftree}
$$

```
Example example2 : |-- [  ] ; [  ]  ;
(> [ ?E{ stepPerp} ;
      atom (p Z) -o atom (p ( S ( S Z)))])
Proof with solveLL.
 simpl...
 decide2 (E{ stepPerp}).
 existential (S Z) .
 tensor [perp(p Z)] [atom (p(S(S Z)))  ].
 decide2 (E{ stepPerp}) .
 existential (Z).
 tensor [perp (p Z) ] [atom (p ( S Z))  ].
Qed.
```

This is a backward chaining proof where the goal $p(2)$ is substituted with $p(1)$ and later with $p(0)$. Note also that the only difference between $\exists FX$ and $\exists FX'$ is the polarity of the atomic formulas. See in Liang and Miller (2009), and Pimentel et al. (2015), a deeper discussion on how the polarity assigned to atoms shapes the proofs in a focused system.

Inversion principles. During the proof of meta-theorems, it is usually necessary to reason under the assumption that a given FLL sequent is provable. For instance, consider the goal:

```
Goal forall (B M : multiset oo) (G:Prop), (|-- B ; M ; (>> (E{ step})) → G.
```

where the unspecified proposition G has to be proved from the fact that $\vdash B : M \Downarrow \exists t.(step\ t)$ is provable. Call this hypothesis H. Since `seq` and `seqN` are inductively defined, Coq's tactic `inversion H` derives all the necessary conditions that should hold to make that sequent provable. However, some spurious cases are generated. In this particular example, `inversion H` generates two subgoals, one of them assuming as hypothesis that the formula $\exists t.(step\ t)$ can lose focusing (which is clearly not the case). Therefore, instead of `inversion`, it is possible to apply the defined tactic `FLLInversion`:

```
Ltac FLLInversion H := match goal with
                       | seqN _ _  _ _ (>> (Some _) ) ⇒ inversion H;subst; [solveF | ]
                       [...]
```

The notation `[solveF| ]` means that the first subgoal (the spurious one) is solved with `solveF` and the second one is left unchanged. Similar cases are considered for the other connectives.

But we can go further due to the focusing discipline. After the introduction of the existential quantifier, the tensor in $p(t)^{\perp} \otimes p(suc(t))$ must be also introduced and focusing continues on $p(t)^{\perp}$ (and one of the initial rules must be applied). Moreover, focusing is lost on $p(suc(t))$ and this atom is necessarily stored in the linear context. All this reasoning is performed by the tactic `FLLInversionAll` that deduces all the necessary hypotheses from the fact that a FLL sequent is provable by repeatedly calling the tactic `FLLInversion`. Coming back to our example, once `FLLInversionAll` is applied, we end up with two subgoals. In both of them, we have as hypotheses `proper t` and $\vdash B : N, p(suc(t)) \Uparrow$ for some N. This means that `FLLInversionAll` showed that there must be a `proper` term `t` due to the existential quantifier. Moreover, the proof has to lose focusing and store the atom $p(suc(t))$ into the linear context. The two subgoals must establish G and, in each case, the following additional hypotheses are available: (1) `Permutation M ((atom (p t)) :: N)` and (2) `In (atom (p t)) B`. These cases result from the *inversion* of the fact that the sequent $\vdash B : M \Downarrow p(t)^{\perp}$ is provable. In the first case, it is assumed that the proof finishes with an application of $[I_1]$ (and then, $p(t) \in M$). In the second subgoal, the proof ended with an application of $[I_2]$ (and $p(t) \in B$).

3.3 Structural properties

Exchange. Using strong induction on the height of the derivation, we show several structural properties for FLL. For instance, we prove that equivalent multisets prove the same formulas (preserving the height of the derivation):

```
Theorem exchangeLCN : Permutation LC LC' → (n |-- CC ; LC ; arrow) → (n |-- CC ; LC' ; arrow).
Theorem exchangeCCN : Permutation CC CC' → (n |-- CC ; LC ; arrow) → (n |-- CC' ; LC ; arrow).
```

From the previous results and using the library `Morphisms`, we are able to easily substitute equivalent multisets during proofs using the tactic `rewrite`. A similar result can be also proved for the list of formulas L in `UP L`. However, in this case, the height of the derivation is not preserved:

```
Theorem ExchangeUp : Permutation L L' → (n |-- B; M ; (> L))  → (|-- B; M ; (> L')).
```

The proof of this theorem requires some lemmas showing the invertibility of the negative connectives. In particular, in a sequent $\vdash \Theta : \Delta \Uparrow L, F, L'$, the formula F can be decomposed or stored at any time during the proof. Here are the cases for $F = \top$ and $F = G \parr G'$:

```
Theorem EquivAuxTop : |-- B ; M ; (> L ++ top :: L')).
Theorem EquivAuxPar : (|-- B ; M ; (> L ++ [F ; G] ++ L')) → (|-- B ; M ; (> L ++ (F | G):: L')).
```

These lemmas are proved by induction on the sum of the complexity of the formulas in L.

$$\dfrac{\vdash \Theta : \Gamma \Uparrow F_c, L \quad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow L}\ [\Uparrow C] \qquad \dfrac{\vdash \Theta : F_c, \Gamma \Uparrow L \quad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow L}\ [\Uparrow LC] \qquad \dfrac{\vdash \Theta, F_c : \Gamma \Uparrow L \quad \vdash \Theta : \cdot \Downarrow !F_c^{\perp}}{\vdash \Theta : \Gamma \Uparrow L}\ [\Uparrow CC]$$

$$\dfrac{\vdash \Theta : F_c, \Gamma \Downarrow F \quad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Downarrow F}\ [\Downarrow LC] \qquad \dfrac{\vdash \Theta, F_c : \Gamma \Downarrow F \quad \vdash \Theta : \cdot \Downarrow !F_c^{\perp}}{\vdash \Theta : \Gamma \Downarrow F}\ [\Downarrow CC]$$

Figure 3. Cut-rules for the system FLL. In rule [$\Downarrow$ *CC*], F_c is not an atom.

Weakening and Contraction. The following results show that weakening and contraction are admissible in the classical context preserving the height of the derivation. Moreover, if a sequent can be proved with a theory `th`, then it can be also proved with a stronger theory `th'`:

```
Theorem weakeningGenN : (n |-- CC ; LC ; arrow) → (n |-- CC' + + CC ; LC ; arrow).
Theorem contractionN  : (n |-- (F :: CC) ; LC ; arrow) → In F CC → (n |-- CC ; LC ; arrow).
Theorem WeakTheoryN   : (forall F, th F → th' F) → (th,n |-- CC ; LC ; arr) → (th',n |-- CC; LC ; arr).
```

4. Metatheory: Cut-elimination for FLL

In Xavier et al. (2017), we mechanized the proof of the cut-elimination theorem for the dyadic system of LL and completeness of the focused system (if a formula has a proof, then it has a focused proof). Hence, as a corollary, we showed the equivalence of all these systems: dyadic + cut, dyadic, and the triadic system. In this paper, we shall not deal with cut-elimination on the dyadic system nor with completeness of focusing. Instead, we propose a set of cut-rules for FLL and prove cut-elimination directly in that system. We present some of the most representative proof transformations. For readability, we have avoided the Coq notation. In the Git repository, the reader can find a PDF with the complete list of steps/cases needed in the proof.

The proposed cut-rules are [$\Uparrow$ *C*], [$\Uparrow$ *LC*], [$\Uparrow$ *CC*], [$\Downarrow$ *LC*], and [$\Downarrow$ *CC*] (see Figure 3). The first three rules deal with unfocused sequents and the last two with focused sequents. Note the side condition in [$\Downarrow$ *CC*] where the cut-formula, from now on denoted as F_c, cannot be an atom. Without this restriction, [$\Downarrow$ *CC*] is not admissible. To see that, consider the sequent $\vdash \top : \cdot \Downarrow A^{\perp}$, which is not provable in FLL. A *non-valid* application of [$\Downarrow$ *CC*] with $F_c = A$ will make that sequent provable. As we shall see, this restriction makes the elimination of [$\Uparrow$ *CC*] more involved.

As usual, the proof of cut-elimination proceeds by double induction on the complexity of the cut-formula and the cut-height, that is, the sum of the premises' heights of the cut-rule. We start in Section 4.1 with the elimination of [$\Uparrow$ *C*]. When $F_c = ?F$ (resp. F_c does not belong to the negative phase), we shall use an application of [$\Uparrow$ *CC*] (resp. [$\Uparrow$ *LC*]) to produce a simpler cut. Section 4.2 shows how to eliminate [$\Uparrow$ *LC*] that, in turn, requires applications of [$\Downarrow$ *LC*] and [$\Uparrow$ *C*] when the list L (in the left premise of the rule [$\Uparrow$ *LC*]) is empty. The elimination of [$\Uparrow$ *CC*] in Section 4.3 is the most involved. Rule [$\Downarrow$ *CC*] is used when the list L is empty and then, F_c cannot an atom. The case when F_c is an atom is eliminated by proving some additional invertibility lemmas. Finally, Section 4.4 shows how to eliminate [$\Downarrow$ *CC*] and [$\Downarrow$ *LC*]. In both cases, we use an application of [$\Uparrow$ *CC*] when the formula F loses focusing. The proofs of this section are in file `CutElimination.v`.

4.1 The cut-rule [$\Uparrow$ *C*]

Assume that F_c is a negative formula. If $F_c = \perp$, the application of [$\Uparrow$ *C*] can be easily eliminated. If the main connective is different from ?, the cut reduction uses induction on the complexity of the formula as exemplified in the case $F \parr G$ below:

$$\dfrac{\dfrac{\overset{[\Sigma_1]}{\vdash \Theta : \Gamma \Uparrow F, G, L}}{\vdash \Theta : \Gamma \Uparrow F \parr G, L} \qquad \dfrac{\overset{[\Sigma_2]}{\vdash \Theta : \Delta_F \Downarrow F^{\perp}} \qquad \overset{[\Sigma_3]}{\vdash \Theta : \Delta_G \Downarrow G^{\perp}}}{\vdash \Theta : \Delta \Downarrow F^{\perp} \otimes G^{\perp}}}{\vdash \Theta : \Gamma, \Delta \Uparrow L}\ [\Uparrow\mathrm{C}] \quad \rightsquigarrow \quad \dfrac{\dfrac{[\Sigma_1] \qquad [\Sigma_2]}{\vdash \Theta : \Gamma, \Delta_F \Uparrow G, L}\ [\Uparrow\mathrm{C}] \qquad [\Sigma_3]}{\vdash \Theta : \Gamma, \Delta \Uparrow L}\ [\Uparrow\mathrm{C}]$$

In the case $F_c = ?F$, F must be stored in the classical context and a reduction similar to that of $F \parr G$ fails. Instead, an application of [$\Uparrow CC$] (that internalizes the storing process) is needed:

$$\dfrac{\dfrac{\vdash \Theta, F : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow ?F, L} \qquad \vdash \Theta : \cdot \Downarrow !\, F^{\perp}}{\vdash \Theta : \Gamma \Uparrow L}\,[\Uparrow\mathrm{C}] \quad \rightsquigarrow \quad \dfrac{\vdash \Theta, F : \Gamma \Uparrow L \qquad \vdash \Theta : \cdot \Downarrow !\, F^{\perp}}{\vdash \Theta : \Gamma \Uparrow L}\,[\Uparrow\mathrm{CC}]$$

When F_c does not belong to the negative phase, an application of [$\Uparrow LC$] is in order:

$$\dfrac{\dfrac{\vdash \Theta : \Gamma, F_c \Uparrow L}{\vdash \Theta : \Gamma \Uparrow F_c, L} \qquad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow L}\,[\Uparrow\mathrm{C}] \quad \rightsquigarrow \quad \dfrac{\vdash \Theta : \Gamma, F_c \Uparrow L \qquad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow L}\,[\Uparrow\mathrm{LC}]$$

4.2 Elimination of [$\Uparrow LC$]

The elimination of [$\Uparrow LC$] when the list L is not empty is trivial: it suffices to permute the cut, thus introducing the first formula in L (and reducing the height of the cut). For instance,

$$\dfrac{\dfrac{\vdash \Theta, F : F_c, \Gamma \Uparrow L}{\vdash \Theta : F_c, \Gamma \Uparrow ?F, L} \qquad [\Sigma]}{\vdash \Theta : \Gamma, \Delta \Uparrow ?F, L}\,[\Uparrow\mathrm{LC}] \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdash \Theta, F : F_c, \Gamma \Uparrow L \qquad \dfrac{[\Sigma]}{\vdash \Theta, F : \Delta \Downarrow F_c^{\perp}}\,[W]}{\vdash \Theta, F : \Gamma, \Delta \Uparrow L}\,[\Uparrow\mathrm{LC}]}{\vdash \Theta : \Gamma, \Delta \Uparrow ?F, L}$$

When the list L is empty, the left premise of [$\Uparrow LC$] must start with a decision rule. If the proof continues by deciding on a formula different from the cut-formula, an application of [$\Downarrow LC$] is needed to reduce the cut-height. Here the case for [D_1]:

$$\dfrac{\dfrac{\vdash \Theta : F_c, \Gamma \Downarrow P}{\vdash \Theta : F_c, P, \Gamma \Uparrow \cdot} \qquad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : P, \Gamma, \Delta \Uparrow \cdot}\,[\Uparrow\mathrm{LC}] \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdash \Theta : F_c, \Gamma \Downarrow P \qquad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Downarrow P}\,[\Downarrow\mathrm{LC}]}{\vdash \Theta : P, \Gamma, \Delta \Uparrow \cdot}$$

If the cut-formula (stored in the linear context) is selected for focusing (using [D_1]), then, depending on the polarity of F_c, we reduce the height of the cut by applying [$\Uparrow C$]:

$$\dfrac{\dfrac{\dfrac{\vdash \Theta : \Gamma \Uparrow F_c}{\vdash \Theta : \Gamma \Downarrow F_c}}{\vdash \Theta : F_c, \Gamma \Uparrow \cdot} \qquad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\,[\Uparrow\mathrm{LC}] \quad \rightsquigarrow \quad \dfrac{\vdash \Theta : \Gamma \Uparrow F_c \qquad \vdash \Theta : \Delta \Downarrow F_c^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\,[\Uparrow\mathrm{C}]$$

$$\dfrac{\dfrac{\vdash \Theta : \Gamma \Downarrow F_c}{\vdash \Theta : F_c, \Gamma \Uparrow \cdot} \qquad \dfrac{\vdash \Theta : \Delta \Uparrow F_c^{\perp}}{\vdash \Theta : \Delta \Downarrow F_c^{\perp}}}{\vdash \Theta : \Gamma \;\; \Delta \Uparrow \cdot}\,[\Uparrow\mathrm{LC}] \quad \rightsquigarrow \quad \dfrac{\vdash \Theta : \Delta \Uparrow F_c^{\perp} \qquad \vdash \Theta : \Gamma \Downarrow F_c}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\,[\Uparrow\mathrm{C}]$$

In the first case, F_c does not belong to the positive phase and the release rule is applied. In the second case, F_c belongs to the positive phase and then, $F_c^{\perp}$ must lose focusing.

4.3 Elimination of [$\Uparrow CC$]

When the list L is not empty, we proceed as in [$\Uparrow LC$]. Otherwise, a decision rule must be used on the left premise of the cut-rule and we have four cases (in [D_2], F_c may be principal or not).

In all of them, we replace an application of $[\Uparrow CC]$ with an application of $[\Downarrow CC]$. Due to the side conditions of $[\Downarrow CC]$, $F_c \neq A$. Here, the case when F_c is principal in $[D_2]$:

$$\dfrac{\dfrac{\vdash \Theta, F_c : \Gamma \Downarrow F_c}{\vdash \Theta, F_c : \Gamma \Uparrow \cdot} \qquad \dfrac{\vdash \Theta : \cdot \Uparrow F_c^{\perp}}{\vdash \Theta : \cdot \Downarrow !\, F_c^{\perp}}}{\vdash \Theta : \Gamma \Uparrow \cdot}\,[\Uparrow\mathrm{CC}] \quad \rightsquigarrow \quad \dfrac{\vdash \Theta : \cdot \Uparrow F_c^{\perp} \qquad \dfrac{\vdash \Theta, F_c : \Gamma \Downarrow F_c \qquad \vdash \Theta : \cdot \Downarrow !\, F_c^{\perp}}{\vdash \Theta : \Gamma \Downarrow F_c}\,[\Downarrow\mathrm{CC}]}{\vdash \Theta : \Gamma \Uparrow \cdot}\,[\Uparrow\mathrm{C}]$$

$[\Uparrow C]$ is applied on a smaller formula and $[\Downarrow CC]$ on a shorter derivation.
The case when $F_c = A$. The proof reduction below:

$$\dfrac{\dfrac{\vdash \Theta, A : \Delta' \Downarrow F}{\vdash \Theta, A : \Delta \Uparrow \cdot} \qquad \vdash \Theta : \cdot \Downarrow !A^{\perp}}{\vdash \Theta : \Delta \Uparrow \cdot}\,[\Uparrow CC] \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\dfrac{\vdash \Gamma, A : \Delta' \Uparrow F \qquad \vdash \Theta : \cdot \Downarrow !A^{\perp}}{\vdash \Theta : \Delta' \Uparrow F}\,[\Uparrow CC]}{\vdash \Theta : \Delta' \Downarrow F}}{\vdash \Theta : \Delta \Uparrow \cdot}$$

works when F is taken either from Δ, from Θ, or from the theory (`th` F) and it loses focusing (including the case $F = !G$). Note that, in the last two cases, $\Delta = \Delta'$. On the other hand, if F cannot lose focusing, the proof relies on the following lemma, showing that there exists a (cut-free) proof of the sequent $\vdash \Theta : \Delta' \Uparrow F$.

Lemma 2 *Consider the derivations below and assume that the rule $[\Uparrow CC]$ is admissible for derivations of height $k < n + m + 1$ where n and m are, respectively, the heights of $[\Sigma]$ and $[\Pi]$. Then, the sequent $\vdash \Theta : \Delta \Uparrow F$ is provable.*

$$\begin{array}{c}[\Sigma]\\ m \vdash \Theta, A : \Delta \Downarrow F\end{array} \qquad \begin{array}{c}[\Pi]\\ \dfrac{\vdash \Theta : \cdot \Uparrow A^{\perp}}{n \vdash \Theta : \cdot \Downarrow !A^{\perp}}\end{array}$$

Note that this lemma can be used to show that the sequent $\vdash \Theta : \Delta \Uparrow \cdot$ is provable when F is a positive formula or $F = A^{\perp}$ (i.e., it cannot lose focusing):

(1) If F was taken from Δ, then we apply the store rule (on $\vdash \Theta : \Delta' \Uparrow F$) and we conclude by noticing that $\Delta', F = \Delta$.

(2) If F was taken from Θ or from the theory `th`, then $\Delta' = \Delta$. We conclude by using the absorption lemmas below (for the classical context and the theory).

We can then complete the elimination of $[\Uparrow CC]$ when the cut-formula is an atom:

$$\{Q \in \Theta\}\ \dfrac{\dfrac{\vdash \Theta, A : \Gamma \Downarrow Q}{\vdash \Theta, A : \Gamma \Uparrow \cdot} \qquad [\Pi]}{\vdash \Theta : \Gamma \Uparrow \cdot}\,[\Uparrow\mathrm{CC}] \qquad \rightsquigarrow \qquad \dfrac{\begin{array}{c}[\Psi]\\ \vdash \Theta : \Gamma \Uparrow Q\end{array}}{\Theta : \Gamma \Uparrow \cdot}\,[\mathrm{Absorption}]$$

where $[\Psi]$ exists due to Lemma 2. Hence, this lemma allows us to handle the case when F_c is an atom. However, the proof of Lemma 2 is far from trivial: it requires proving some of the invertibility lemmas needed by Andreoli in his proof of completeness. These lemmas were already formalized in Xavier et al. (2017). Roughly speaking, these results show that applications of positive rules can be switched:

Absorption classical context: If $\vdash \Gamma, F : \Delta \Uparrow F, L$ then $\vdash \Gamma, F : \Delta \Uparrow L$
Absorption theory: If $(th\ F)$, F is not a literal and $\vdash \Gamma : \Delta \Uparrow F, L$ then $\vdash \Gamma, : \Delta \Uparrow L$
Absorption atom: If $(th\ A^{\perp})$ and $\vdash \Gamma : A^{\perp}, \Delta \Updownarrow$ then $\vdash \Gamma : \Delta \Updownarrow$
Inversion of $\otimes$: if $\vdash \Gamma : \Delta \Uparrow F, L$ and $\vdash \Gamma : \Delta' \Uparrow F', L'$ then $\vdash \Gamma : \Delta, \Delta', F \otimes F' \Uparrow L, L'$

Inversion of $\oplus_i$: If $\vdash \Gamma : \Delta \Uparrow F_i, L$ then $\vdash \Gamma : \Delta, F_1 \oplus F_2 \Uparrow L$
Inversion of $\exists$: If $\vdash \Gamma : \Delta \Uparrow F[t/x], L$ then $\vdash \Gamma : \Delta, \exists x.F \Uparrow L$.

Proof of Lemma 2. We proceed by induction on m. If $m = 0$ then either F is the unit 1 or a negated atom. The case $F = 1$ is trivial. If F is a negated atom:

- If $F = B^\perp$ and $A \neq B$ then either $B \in \Gamma$ or $\Delta' = B$ and the result is easy.
- If $F = A^\perp$ we consider two cases: (1) if $\Delta = A$ the result is trivial (by storing $A^\perp$ to later focus on it); and (2), if $\Delta = \cdot$ then the result holds by using the derivation $[\Pi]$.

Now consider the case $m > 0$. We proceed by case analysis on the last rule applied in $[\Sigma]$. If the *release* rule is applied, we can use $[\Uparrow CC]$ to exhibit a shorter derivation $(n + m - 1)$:

$$\dfrac{\overset{[\Sigma']}{\vdash \Theta, A : \Delta \Uparrow F}}{\vdash \Theta, A : \Delta \Downarrow F} \quad \rightsquigarrow \quad \dfrac{m-1 \vdash \Theta, A : \Delta \Uparrow F \qquad \dfrac{\overset{[\Pi]}{\vdash \Theta : \cdot \Uparrow A^\perp}}{n \vdash \Theta : \cdot \Downarrow !A^\perp}}{\vdash \Theta : \Delta \Uparrow F}\ [\Uparrow CC]$$

If $F = !G$, then focusing is lost and we use $[\Uparrow CC]$ as above to obtain a proof of $\vdash \Theta : \cdot \Uparrow G$. Then, we obtain a proof of $\vdash \Theta : \cdot \Uparrow !G$ by using $[R \Uparrow]$ to later focus on $!G$.

If $[\Sigma]$ starts with $[\otimes]$ and $F = G_1 \otimes G_2$, by induction, we have proofs of both: $\vdash \Theta : \Delta_1 \Uparrow G_1$ and $\vdash \Theta : \Delta_2 \Uparrow G_2$ where $\Delta = \Delta_1 \cup \Delta_2$. By the inversion lemma of $\otimes$, $\vdash \Gamma : \Delta_1, \Delta_2, G_1 \otimes G_2 \Uparrow \cdot$. Using the store rule, we conclude $\vdash \Theta : \Delta_1, \Delta_2 \Uparrow G_1 \otimes G_2$ as needed. The case $F = G_1 \oplus G_2$ (resp. $F = \exists x.G$) follows similarly, by induction and using the inversion lemma for $\oplus$ (resp. $\exists$). □

4.4 Elimination of $[\Downarrow CC]$ and $[\Downarrow LC]$

The procedure for these two rules is simple since, in the left premise, the focused formula is necessarily principal. Then, it is possible to permute the cut to obtain a shorter derivation. If the proof changes polarity, then an application of $[\Uparrow CC]$ (or $[\Uparrow LC]$ in the case of $[\Downarrow LC]$) is in order:

$$\dfrac{\dfrac{\vdash \Theta : F_c, \Gamma \Uparrow P}{\vdash \Theta : F_c, \Gamma \Downarrow P} \qquad [\Sigma]}{\vdash \Theta : \Gamma, \Delta \Downarrow P}\ [\Downarrow\text{LC}] \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\vdash \Theta : H, \Gamma \Uparrow P \qquad [\Sigma]}{\vdash \Theta : \Gamma, \Delta \Uparrow P}\ [\Uparrow\text{LC}]}{\vdash \Theta : \Gamma, \Delta \Downarrow P}$$

4.5 The main result

Now we can prove the main result of this section. We proceed by mutual induction on the five rules and using the reductions presented in the preceding sections.

Theorem (Cut-elimination). *The rules $[\Uparrow C]$, $[\Uparrow LC]$, $[\Uparrow CC]$, $[\Downarrow LC]$, $[\Downarrow CC]$ are admissible.*

We have also proved for FLL *identity elimination*, that is, for any well-formed formula F and context Θ, the sequent $\vdash \Theta : \cdot \Uparrow F, F^\perp$ is provable. Since we are dealing with a focused system, the proof of this theorem is not immediate (as in the case of unfocused systems) and some invertibility lemmas are required. The reader can find the details in `IdElimination.v`.

Before closing this section, we note that rules $[\Downarrow LC]$ and $[\Uparrow LC]$ allow us to introduce negative formulas into the linear context, while the system FLL always decomposes such formulas during the negative phase. We can add as a side condition that F_c is a positive formula or a literal in both rules. In that case, our proof remains unchanged. To see that, note that $[\Uparrow LC]$ is used in the elimination of $[\Uparrow C]$ when, precisely, F_c does not belong to the negative phase. $[\Downarrow LC]$ is used in the elimination of $[\Uparrow LC]$ and vice versa. If both rules have the same side condition, the reductions presented here continue to be valid.

5. Applications

It is possible to use our framework to reason about LL specifications. For instance, we may encode the states of a system as atomic predicates s,t, etc., and define a theory storing formulas of the form $F = s^{\perp} \otimes t$. When focusing on F, the atom s is consumed and t is produced, thus faithfully encoding the state transition $s \to t$. Hence, we can prove in FLL reachability properties such as *the state s'' is reachable from* s. When needed, the cut-rule in FLL can be used to show that some state s'' is reachable from s and s' is reachable from s''. Specifying the syntax (for states) and transition rules is easy. As an example, the directory `OL/TSystem` contains the specification of the transition rules for the biochemical system studied in Despeyroux et al. (2018) along with the proofs of some of its properties. The state of the system takes the form $T(t), C_1(\vec{x_1}), \cdots, C_n(\vec{x_n})$ where the atom $T(t)$ represents the current time unit and $C_i(\vec{x_i})$ the state of a cell. The 171 transition rules of the model consume and produce such atoms to model a cell's life cycle. The resulting proofs are relatively short due to the automatic tactics implemented.

We can also encode the inference rules of other logical systems as LL theories for different purposes. This is a more interesting application, since the syntax of the OL usually includes binders (e.g., the first-order quantifiers). The present section explores these encodings and their application in: (1) proving cut-elimination of the OL by relying on the metatheory of FLL; (2) formalizing the proof that HyLL is not more expressive than LL; and (3) showing the equivalence between the natural deduction and sequent calculus systems for minimal logic. These results are certainly a compelling application of the infrastructure provided by our framework to reason about LL specifications.

We proceed as follows. Section 5.1 describes how to encode the syntax of an OL. As a running example, we shall use the system LK for first-order classical logic. An LK sequent of the form $\Gamma \vdash \Delta$ will be encoded as the FLL sequent $\vdash \lfloor\Gamma\rfloor, \lceil\Delta\rceil : \cdot \Uparrow$ where $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ are predicates that we introduce in Section 5.2. The inference rules of LK will be encoded as an LL theory and we shall prove that this encoding is adequate. In Section 5.3, we formalize the notion of cut-coherence (Miller and Pimentel, 2013), a necessary condition for cut-elimination of the OL specified as an LL theory. Cut-coherence states that the right and left inference rules of the OL are self-dual and Theorem 4.5 allows for proving the OL's cut-elimination theorem. We shall see that this result applies for different systems that can be adequately encoded in LL including classical (LK), minimal (LM), and intuitionistic (LJ) logics and variants of LL (Sections 5.3 and 5.4).

In Section 5.5, we formalize a result from Chaudhuri et al. (2019a). Namely, we prove that the inference rules of HyLL can be adequately encoded as LL theories. This shows that HyLL's judgments based on worlds ($F@w$, F holds at world w) do not increase the expressiveness power of LL. For this encoding, we have to deal with binders on first-order terms and worlds, and both are uniformly handled in Hybrid. We also present the second encoding of HyLL that, although adequate only at the level of proofs, allows us to establish cut-coherence and hence, cut-elimination for HyLL. Finally, Section 5.6 shows that it is possible to prove the (LL) equivalence between the encodings of two different logical systems, thus obtaining mutual relative completeness results. In particular, we prove that the LL encoding of the right (resp. left) rules of LM are equivalent to the introduction (resp. elimination) rules of a natural deduction system for minimal logic. This shows that both systems prove the same formulas.

The encodings presented here for LK, LJ, and LM are not the same as the ones introduced in Miller and Pimentel (2013). Our encodings can be proved to be adequate at the level of derivations (Nigam and Miller, 2010): there is a 1-1 relation between the set of derivations in the OL and focused derivations in FLL. In *op. cit.*, the encodings are adequate only at the level of proofs: an OL sequent is provable iff its encoding is provable in FLL. This weaker notion of adequacy is enough to show that cut-coherence implies cut-elimination for the OL. However, it requires proving extra meta-theorems showing that a non-adequate application of the encoded rule cannot prove more sequents (some examples in Section 5.5). The cut-coherence result for HyLL in Sec. 5.5 is new.

5.1 Specifying the syntax of the OL

OL formulas are built from terms, atomic propositions, and connectives. The file `OLSyntax.v` defines the following type class that makes the definition of the OL syntax easier:

```
Class OLSyntax := {
    OLType:Set;          (* OL terms *)               constants : Set ;   (* 0-ary connectives *)
    connectives : Set ;  (* binary connectives *)     quantifiers : Set}.  (* quantifiers *)
```

The first field determines the type for building terms at the object level. This set can be Coq's `Empty_set` in the case of propositional OLs. The other fields define the constants used to represent the OL connectives, quantifiers, and units (0-ary connectives). For example, the following constants determine the syntactic elements of first-order classical logic:

```
Inductive Constants := TT | FF .          (* true and false units *)
Inductive Connectives := AND | OR | IMPL . (* conjunction, disjunction and implication *)
Inductive Quantifiers := ALL | EX .       (* universal and existential quantifiers *)
```

These definitions can be used to instantiate the class `OLSyntax`:

```
Instance SimpleOLSig : OLSyntax:= {|
    OLType := nat; (* natural numbers will be the terms of the OL *)
    constants := Constants          ;    connectives := Connectives    ;    quantifiers := Quantifiers |}.
```

We then inductively define the type `Econ` that will instantiate the type `con` of Hybrid to build the actual constants of the OL syntax of type `expr Econ` (see Section 2):

```
Inductive Econ: Set :=
  | oo_term : OLType → Econ .     (* terms *)        | oo_atom : nat → Econ          (* atomic props *)
  | oo_cons : constants → Econ    (* units *)
  | oo_bin : connectives → Econ   (* connectives *) | oo_q : quantifiers → Econ     (* quantifiers *)
```

In `oo_atom`, the parameter `n:nat` is the identifier of the atomic proposition (at the object level). With these definitions, it is possible to provide more suitable "constructors" to work with:

```
Definition uexp : Set := expr Econ. (* uexp is the type of OL terms, atoms and formulas *)
Definition t_term (t:OLType) :=   (CON (oo_term t))  .                (* terms *)
Definition t_atom (id:nat) (A:uexp) := APP (CON (oo_atom id)) A.      (* atoms *)
Definition t_cons (lab :constants) := CON (oo_cons lab) .             (* constants *)
Definition t_bin (lab : connectives) : uexp → uexp → uexp :=          (*  connectives *)
    fun M1:uexp ⇒ fun M2:uexp ⇒ (APP (APP (CON (oo_bin lab )) M1) M2).
Definition t_quant (lab : quantifiers) : (uexp → uexp) → uexp :=      (* quantifiers *)
    fun M:uexp→ uexp ⇒ (APP (CON (oo_q lab)) (lambda M)).
```

Note that these definitions are the only place where the constructors of the `expr` type (`con`, `APP` and `lambda`) are seen explicitly. Once the new constructors above are defined, we use only these. For instance, at the object level, the conjunction $A \wedge B$ is written as `t_bin AND A B` and the universal quantified formula $\forall x.FX$ as `t_quant ALL FX`.

There is also a predicate establishing when a `uexp` term is indeed a well-formed OL formula:

```
(* OL terms can be built only with t_term *)
Inductive isOLTerm : uexp → Prop := | isOLTermT : forall t, isOLTerm (t_term t).
(* Formulas can be atomic propositions or formulas built from the OL connectives *)
Inductive isOLFormula : uexp → Prop :=
| isFAtom : forall t id , isOLTerm t → isOLFormula (t_atom id t) (* atoms *)
| isFQ : forall lab (FX : uexp → uexp), (* quantifiers *)
      uniform FX → (forall (t:uexp), proper t → isOLFormula (FX t)) → isOLFormula (t_quant lab FX) .
```

We omit the definitions for constants and connectives which are similar. Note that the quantified formula `t_quant ALL FX` is a valid OL formula whenever `FX` is `uniform` and, for all `proper` term t, the resulting formula $FX\ t$ is also a valid formula.

It is important to show that both syntax and inference rules of an OL are adequately represented in Hybrid. Adequacy for inference rules is stated and proved in Section 5.2. Adequacy of syntax encoding, also called *representational adequacy*, is discussed for the lambda calculus as an OL in Hybrid in Ambler et al. (2002) and proved in detail in Crole (2011), adequacy for a fragment of a functional programming language known as Mini-ML is proved in Felty and Momigliano (2012), and adequacy for a quantum programming language whose core is a linear lambda calculus is discussed in detail in Mahmoud and Felty (2019). The OLs described here are similar to these others in the sense that the quantifiers in this paper are represented in the same manner as the lambda operator of the lambda calculus in the cited papers, and thus such adequacy results can be adapted fairly directly. In particular, the predicates `uniform` and `proper` are important for this task.

In addition, proving representational adequacy requires defining an encoding function between OL terms and their representation in Hybrid, and showing that this function is a bijection; for this task also, previous proofs can be adapted directly. Here the `isOLFormula` is important for this proof. It is also important for reasoning about the OL. In particular, under the assumption that `isOLFormula F` holds, we are able to reason inductively on the structure of F.

We define the size of OL formulas with the inductively defined predicate `lengthUexp` satisfying the expected properties: `lengthUexp A 1` if A is a (well-formed) OL atomic formula or a constant; if `lengthUexp F n` and `lengthUexp G m`, then `lengthUexp (t_bin c F G) (n +m +1)` for any connective `c`, etc.

Finally, in file `OLSyntax.v`, we define the LL signature containing the predicates `up` and `down` to represent, respectively, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$. These predicates will be used in the next section:

```
Inductive atm : Set := up : uexp → atm | down : uexp → atm.
```

```
Inductive atm : Set := up : uexp → atm | down : uexp → atm .
```

5.2 OL inference rules as LL theories and adequacy

In Miller and Pimentel (2013), LL was used as a logical framework for specifying a number of logical systems. The idea is to use the predicates $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ for identifying objects that appear on the left or on the right side of the sequents in the OL. Hence, for instance, object-level sequents of the form $B_1, \ldots, B_n \vdash C_1, \ldots, C_m$ (where $n, m \geq 0$) are specified as the multiset of atomic LL formulas $\lfloor B_1 \rfloor, \ldots, \lfloor B_n \rfloor, \lceil C_1 \rceil, \ldots, \lceil C_m \rceil$. As a mnemonic, formulas on the (**L**)eft side of object-level sequents are encoded with the predicate starting with $\lfloor$. Depending on the presence of the structural rules of weakening and contraction in the OL, the above formulas can be stored in the linear or in the classical context of FLL. For instance, in LK, the structural rules apply on both sides of the sequent and the LK sequent $\Gamma \vdash \Delta$ will be encoded as the FLL sequent $\vdash \lfloor \Gamma \rfloor, \lceil \Delta \rceil : \cdot \Uparrow$ where $\lfloor \Gamma \rfloor = \{\lfloor F_i \rfloor \mid F_i \in \Gamma\}$ (similarly for $\lceil \Delta \rceil$).

Inference rules of the OL are specified as rewriting clauses that replace the active formula in the conclusion of the rule by the resulting formulas in the premises. The LL connectives indicate how these object-level formulas are connected: contexts are copied (&) or split ($\otimes$), in different inference rules ($\oplus$) or in the same sequent ($\parr$).

Consider for instance the following rules for conjunction in classical logic:

$$\dfrac{\Gamma, F \longrightarrow \Delta}{\Gamma, F \wedge G \longrightarrow \Delta}\wedge_{L1} \qquad \dfrac{\Gamma, G \longrightarrow \Delta}{\Gamma, F \wedge G \longrightarrow \Delta}\wedge_{L2} \qquad \dfrac{\Gamma \longrightarrow F, \Delta \quad \Gamma \longrightarrow G, \Delta}{\Gamma \longrightarrow F \wedge G, \Delta}\wedge_R$$

These rules yield the following LL formulas where $\lfloor F \wedge G \rfloor^\perp$ and $\lceil F \wedge G \rceil^\perp$ are the heads of the clauses (the principal formula in the conclusion of each inference rule):

$$\wedge_L : \exists F, G.(\lfloor F \wedge G \rfloor^\perp \otimes (?\lfloor F \rfloor \oplus ?\lfloor G \rfloor)) \qquad \wedge_R : \exists F, G.(\lceil F \wedge G \rceil^\perp \otimes (?\lceil F \rceil \& ?\lceil G \rceil))$$

The formulas resulting from the encoding of the logical rules are not arbitrary. In fact, those formulas are *bipoles* (Andreoli, 1992) that, when focused on, are completely decomposed into their atomic subformulas. Below are the resulting derivations when focusing on $\wedge_L$ and $\wedge_R$:

$$\dfrac{\dfrac{\dfrac{}{\vdash \Theta : \cdot \Downarrow \lfloor F \wedge G \rfloor^\perp}[I_2] \quad \dfrac{\dfrac{\vdash \Theta, \lfloor F \rfloor : \cdot \Uparrow \cdot}{\vdash \Theta : \cdot \Downarrow ?\lfloor F \rfloor}[R \Downarrow ?]}{\vdash \Theta : \cdot \Downarrow ?\lfloor F \rfloor \oplus ?\lfloor G \rfloor}[\oplus_l]}{\vdash \Theta : \cdot \Downarrow \exists F, G.(\lfloor F \wedge G \rfloor^\perp \otimes (?\lfloor F \rfloor \oplus ?\lfloor G \rfloor))}[\exists\exists\otimes]}{\vdash \Theta : \cdot \Uparrow \cdot}[D_3] \qquad \dfrac{\dfrac{\dfrac{}{\vdash \Theta : \cdot \Downarrow \lceil F \wedge G \rceil^\perp}[I] \quad \dfrac{\vdash \Theta, \lceil F \rceil : \cdot \Uparrow \cdot \quad \vdash \Theta, \lceil G \rceil : \cdot \Uparrow \cdot}{\vdash \Theta : \cdot \Downarrow (\lceil F \rceil \& \lceil G \rceil)}[R \Downarrow \& R \Uparrow]}{\vdash \Theta : \cdot \Downarrow \exists F, G.(\lceil F \wedge G \rceil^\perp \otimes (\lceil F \rceil \& \lceil G \rceil))}[\exists\exists\otimes]}{\vdash \Theta : \cdot \Uparrow \cdot}[D_3]$$

Consider the derivation on the left that starts by focusing on the formula $\wedge_L$ stored in the theory `th`. The connectives $\exists$, $\otimes$ and $\oplus$ must be all introduced during the positive phase. In the left branch of this derivation, the initial rule $[I_2]$ must be applied and, necessarily, $\lfloor F \wedge G \rfloor \in \Theta$. Focusing is lost on the formula $?\lfloor F \rfloor$ and $\lfloor F \rfloor$ is stored in the classical context during the negative phase. What we observe, bottom-up, is that the active formula $F \wedge G$ is decomposed and the whole positive phase (after the resulting negative phase) ends by storing the atom $\lfloor F \rfloor$ into the classical context. This derivation reflects exactly an application of the rule $\wedge_{L1}$ at the object level. Similarly, if instead of $[\oplus_l]$ we apply $[\oplus_r]$, the atom $\lfloor G \rfloor$ is stored, thus reflecting the behavior of $\wedge_{L2}$.

Now consider the second derivation (on the right) that starts by focusing on $\wedge_R$. Note that this derivation ends up with two premises, corresponding exactly to the two premises of the rule $\wedge_R$.

Let us consider an alternative presentation and encoding for conjunction:

$$\dfrac{\Gamma, F, G \longrightarrow \Delta}{\Gamma, F \wedge G \longrightarrow \Delta}\wedge_L \qquad \dfrac{\Gamma \longrightarrow F, \Delta \quad \Gamma \longrightarrow G, \Delta}{\Gamma \longrightarrow F \wedge G, \Delta}\wedge_R \qquad \begin{array}{l} \wedge_L : \exists F, G.(\lfloor F \wedge G \rfloor^\perp \otimes (?\lfloor F \rfloor \parr ?\lfloor G \rfloor)) \\ \wedge_R : \exists F, G.(\lceil F \wedge G \rceil^\perp \otimes (?\lceil F \rceil \otimes ?\lceil G \rceil)) \end{array}$$

In $\wedge_L$, focusing is lost on the formula $?\lfloor F \rfloor \parr ?\lfloor G \rfloor$ and both, $\lfloor F \rfloor$ and $\lfloor G \rfloor$ are stored into the classical context. Since the linear context in this encoding is always empty, any of the two conjunctions of LL adequately encode the behavior of the rule $\wedge_R$. However, we prefer the use of $\otimes$ instead of & since, as we shall see, this allows us to prove that the left/right rules are self-dual.

The OL theory. LL formulas specifying OL inference rules take the form $\exists \overline{F}(H \otimes B)$ where H is the head of the rule and B its body. The head H can be easily inferred, while the body B is specific for each connective/rule. Hence, the following class is defined:

```
Class OORules := { rulesCte : constants → ruleCte      ;
                   rulesBin : connectives → ruleBin  ;    rulesQ :   quantifiers → ruleQ }.
```

where `ruleCte`, `ruleBin`, and `ruleQ` are records specifying the body of the right and left rules for each connective. Let us exemplify the situation with `rulesBin`; the other cases are similar:

```
Record ruleBin := {  ru_rightBody : uexp → oo; (* body of the right introduction rule *)
                     ru_leftBody : uexp → oo}. (* body of the left introduction rule *)
```

The specification of LK requires a term of type `ruleBin` for each binary connective:

```
Definition rulesBC (c:connectives) : ruleBin :=
match c with     | AND ⇒ {|   ru_rightBody := fun F G ⇒ (? u|F|) ** (? u|G|) ;
                              ru_leftBody  := fun F G ⇒ (? d|F|) | (? d|G|) |}
                 | OR ⇒ {|    ru_rightBody := fun F G ⇒ (? u|F|) | (? u|G|) ;
                              ru_leftBody  := fun F G ⇒ (? d|F|) ** (? d|G|) |}
                 | IMPL := {| ru_rightBody := fun F G ⇒ (? d|F|) | (? u|G|);
                              ru_leftBody  := fun F G ⇒ (? u|F|) ** (? d|G|) |}   end.
```

where `u|F|` (resp. `d|F|`) denotes the atom $\lceil F \rceil$ (resp. $\lfloor F \rfloor$).

The encoding of the rule is obtained by adding the head of the rule to its body. This is the purpose of the following definitions:

```
Definition makeLRuleBin (c : connectives) := fun F G ⇒ d^| t_bin c F G| ** (rulesBin c).(rb_leftBody) F G .
Definition makeRRuleBin (c : connectives) := fun F G ⇒ u^| t_bin c F G| ** (rulesBin c).(rb_rightBody) F G.
```

where `d^| F|` denotes $\lfloor F \rfloor^\perp$. Similar constructors for constants and quantifiers are defined. Using them, it is possible to build the LL theory corresponding to the encoding of all the inference rules:

```
Inductive buildTheory : oo → Prop :=
  | bcteR : forall C, isOLFormula (t_cons C)            → buildTheory (makeRRuleConstant C)
  | bcteL : forall C, isOLFormula (t_cons C)            → buildTheory (makeLRuleConstant C)
  | bconnR : forall C F G, isOLFormula ( t_bin C F G)   → buildTheory (makeRRuleBin C F G)
  | bconnL : forall C F G, isOLFormula ( t_bin C F G)   → buildTheory (makeLRuleBin C F G)
  | bQconnR : forall C FX, isOLFormula (t_quant C FX)   → buildTheory (makeRRuleQ C FX)
  | bQconnL : forall C FX, isOLFormula (t_quant C FX)   → buildTheory (makeLRuleQ C FX) .
```

Notice the inclusion of the well-formed predicate (`isOLFormula`): in order to focus on the formula (`makeLRuleBin AND F G`), encoding the rule $\wedge_L$, both F and G must be well-formed OL formulas.

In addition to the inference rules of the OL, we require an LL formula specifying the application of the initial rule at the object level:

```
Definition RINIT F := ( u^| F| ) ** (d^| F| ) . (* i.e., ⌈F⌉⊥ ⊗ ⌊F⌋⊥ *)
Inductive OLTheory : oo → Prop := (* inference rules + init *)
 | ooth_rules : forall OO, buildTheory OO   → OLTheory OO
 | ooth_init : forall OO, isOLFormula OO    → OLTheory (RINIT OO) .
```

Note that in `RINIT`, focus cannot be lost on $\lceil F \rceil^\perp$ or on $\lfloor F \rfloor^\perp$. This adequately encodes the initial rule at the object level: F must be present on both, the left ($\lfloor F \rfloor$) and right ($\lceil F \rceil$) side of the OL sequent.

Adequacy. The provability relation $\cdot \vdash \cdot$ of LK can be specified as an inductive predicate in Coq:

```
Inductive LKSeq : multiset uexp → multiset uexp → Prop :=
| LKinit : forall Gamma Delta F, LKSeq (F:: Gamma) (F:: Delta)
| LKAndL : forall Gamma Delta F G, LKSeq (F :: G :: Gamma) Delta → LKSeq ( (t_bin AND F G) :: Gamma) Delta
[...]
```

and we can prove that the LL encoding is sound and complete:

```
Theorem Adequacy: isOLFormulaL Gamma → isOLFormulaL Delta →
(LKSeq Gamma Delta) ↔ seq OLTheory (LEncode Gamma ++ REncode Delta ) [ ] (> [])).
```

Given a list of (OL) formulas Γ, `LEncode Gamma` (resp. `REncode Gamma`) returns $\lfloor\Gamma\rfloor$ (resp. $\lceil\Gamma\rceil$). The predicate `isOLFormulaL Gamma` states that all the formulas in `Gamma` satisfy `isOLFormula`. The proof of this theorem is almost straightforward by using the tactics of the framework (see `LK.v`). For soundness ($\rightarrow$), we proceed by induction on the fact that the sequent $\Gamma \vdash \Delta$ is provable. In each case, it suffices to apply the corresponding LL rule stored in `OLTheory`. For completeness (<-), we have as hypothesis that the sequent $\vdash \lfloor\Gamma\rfloor, \lceil\Delta\rceil : \cdot \Uparrow$ is provable. Since $\lfloor\Gamma\rfloor$ and $\lceil\Delta\rceil$ contain only atomic formulas, such a proof must start with $[D_3]$ (thus focusing on one of the formulas in `OLTheory`). In each case, by using the tactic `FLLInversionAll`, we can derive all the consequences from the fact that such a sequent is provable and the result follows by induction.

5.3 Cut-coherence and cut-elimination

The rule $\dfrac{\Gamma \vdash F, \Delta \quad \Gamma, F \vdash \Delta}{\Gamma \vdash \Delta}\, cut$ in LK can be adequately encoded as $\exists F.(?\lceil F\rceil \otimes ?\lfloor F\rfloor)$. Note that focusing is lost on both $?\lceil F\rceil$ and $?\lfloor F\rfloor$, thus producing two premises: one where F is added into the left side of the sequent and the other where F is added into the right side. The following theory

```
Definition RCUT F   := ?u|F|  **  ?d|F|.
Inductive OLTheoryCut (n:nat) : oo → Prop := (* OLTheory + Cuts on formulas of size at most n *)
  | oothc_theory : forall OO, OLTheory OO → OLTheoryCut n OO
  | oothc_cutn : forall F m ,  isOLFormula F → lengthUexp F m → m <= n → OLTheoryCut n (RCUT F).
```

adds to `OLTheory` the rule `RCUT` applied on formulas of size (`lengthUexp`) at most n. Since there are no OL formulas of size 0, clearly the theory `(OLTheoryCut 0)` is equivalent to `OLTheory`. Moreover, the theory `(OLTheoryCut n)` is stronger than the theory `(OLTheoryCut m)` whenever $m \leq n$.

Cut-coherence. Let B_R and B_L be the instances (after introducing the existential variables) of the right and left body of the encodings of the logical rules of a given connective. Miller and Pimentel (2013) define cut-coherence as the property that, using a theory containing `RCUT`, the sequent $\vdash \cdot : \cdot \Uparrow B_R^{\perp} \parr B_L^{\perp}$ is provable. In our case, the definition of cut-coherence needs to be more specific regarding the size n of the (OL) cut-formula. This is needed in order to consider the weakest theory (`OLTheoryCut n`) that makes the sequent above provable. We have a definition for each kind of connective, for instance,

```
Definition CutCoherenceCte (R: ruleCte) :=
 EmptyTheory |-- [] ;  []  ;  (> [dual (  R.( rc_rightBody) ) ; dual (R.( rc_leftBody) )]).
Definition CutCoherenceBin (R: ruleBin) :=
forall F G n m,   lengthUexp F n → lengthUexp G m → isOLFormula F → isOLFormula G→
 (CutRuleN (max n m)) |-- []  ;  []  ;  (> [dual (  R.( rb_rightBody) F G) ; dual (R.( rb_leftBody) F G)]).
```

For units, `RCUT` is not needed for proving the duality (`EmptyTheory F` does not hold for any F). For a binary connective $F \star G$, it is enough to consider a theory able to prove both the duality for F and G. We then say that a system is cut-coherent if all its rules satisfy the cut-coherence condition.

Proving cut-coherence is rather simple. Consider for instance the case of conjunction in LK. After the negative phase, we need to prove the following sequent $\vdash : (!\lfloor F\rfloor^{\perp} \otimes !\lfloor G\rfloor), !\lceil F\rceil^{\perp}, !\lceil G\rceil^{\perp} \Uparrow$. By focusing on `RCUT F`, we can replace $!\lceil F\rceil^{\perp}$ with $?\lfloor F\rfloor$ (similarly for G). Hence, the goal is to prove $\vdash \lfloor F\rfloor, \lfloor G\rfloor : !\lfloor F\rfloor^{\perp} \otimes !\lfloor G\rfloor \Uparrow$ which is easy by focusing on $!\lfloor F\rfloor^{\perp} \otimes !\lfloor G\rfloor$. Note that `RCUT` was applied only on subformulas of $F \wedge G$.

Regarding the first-order OL quantifiers, consider the following LL formulas encoding the right and left inference rules for the universal quantified formula $fx.FX$:

$$\forall_R : \exists\, FX.(\lceil fx.FX \rceil^{\perp} \otimes \forall x.?\lceil FX\ x \rceil) \quad \forall_L : \exists\, FX.(\lfloor fx.FX \rfloor^{\perp} \otimes \exists x.?\lfloor FX\ x \rfloor)$$

As expected, the right rule uses the LL universal quantifier to create a fresh variable x and stores $\lceil FXx \rceil$ into the classical context. On the other side, the left rule uses the LL existential quantifier and, for a given term t, $\lfloor FX\ t \rfloor$ is stored. The bodies of these rules are clearly cut-coherent. In order to prove this result, we have to show that given two (OL) `proper` terms x and y, if the size of $(FX\ x)$ is n, then the size of $(FX\ y)$ is also n. This is indeed the case since FX is `uniform` (and it cannot return different formulas depending on its parameters). However, proving this fact is not doable with the infrastructure we have and the cut-coherence of these rules relies on the following admitted axiom: `Axiom OLSize: uniform FX → proper t → proper t' → lengthUexp (FX t) n → lengthUexp (FX t') n.`

The OL cut-elimination theorem. Now we are ready to formalize the cut-elimination procedure for OLs satisfying cut-coherence. We start with the following theorem:

```
Theorem OLCutStep: isOLFormula FCut → lengthUexp FCut n' → n'<=n → IsPositiveAtomFormulaL Gamma →
   (h1, OLTheory |-- (d|FCut| :: Gamma);[];(> [])) → (h2, OLTheory |-- (u|FCut| :: Gamma);[];(> [])) →
   (OLTheoryCut (pred n) ) |-- Gamma ; [] ;(> []) .
```

In words, assuming that: the cut-formula (`FCut`) is a well-formed OL formula of size $\leq n$; Γ contains only atoms of the shape $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ (`IsPositiveAtomFormulaL Gamma`); there are cut-free proofs (`OLTheory`) of the sequents $\vdash \lfloor FCut \rfloor, \Gamma : \cdot \Uparrow \cdot$ and $\vdash \lceil FCut \rceil, \Gamma : \cdot \Uparrow \cdot$ of height, respectively, h_1 and h_2; then, there is a proof of the sequent $\vdash \Gamma : \cdot \Uparrow \cdot$ with the theory that considers cuts on formulas strictly smaller than $FCut$ (`pred n`).

This proof proceeds by induction on $h_1 + h_2$. Note that the sequents in the hypotheses must start by focusing on one of the formulas in `OLTheory`. Hence, we are in the following situation:

$$(1) \vdash \lfloor FCut \rfloor, \Gamma : \cdot \Downarrow R_1 \qquad (2) \vdash \lceil FCut \rceil, \Gamma : \Downarrow R_2$$

where $R_1 = H_1 \otimes B_1$ and $R_2 = H_2 \otimes B_2$. If $H_1 = \lfloor FCut \rfloor^{\perp}$ and $H_2 = \lceil FCut \rceil^{\perp}$ we are in the case where the cut formula is principal in both premises. Hence, after decomposing the heads of the rules, we are in the following situation:

$$(1') \vdash \lfloor FCut \rfloor, \Gamma : \cdot \Downarrow B_1 \qquad (2') \vdash \lceil FCut \rceil, \Gamma : \Downarrow B_2$$

We know that decomposing B_1 and B_2 necessarily ends up generating zero, one or two premises, where more atoms of the form $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ are added into the classical context. Consider that one of these premises is $\vdash \lfloor FCut \rfloor, \Gamma, \Gamma' : \cdot \Uparrow$. By induction (on a shorter derivation), it must be the case that $\vdash \Gamma, \Gamma' : \cdot \Uparrow$ is provable and then, the sequent $\vdash \Gamma : \cdot \Downarrow B_1$ is also provable. By applying the same reasoning on (2'), we can show that

$$(1'') \vdash \Gamma : \cdot \Downarrow B_1 \qquad (2'') \vdash \Gamma : \Downarrow B_2$$

Both sequents are provable in the theory `OLTheoryCut (pred n)` . Due to cut-coherence, the sequent $\mathcal{S} = \vdash \cdot : \cdot \Uparrow B_1^{\perp}, B_2^{\perp}$ is provable and, by weakening, the sequent $\mathcal{S}' = \vdash \Gamma : \cdot \Uparrow B_1^{\perp}, B_2^{\perp}$ is also provable. We conclude by using the cut-rule of FLL twice: cutting $\mathcal{S}'$ with $(1'')$ and then cutting the resulting sequent with $(2'')$. Hence, the principal cases of cuts at the object level are all eliminated by cuts at the meta-level, which, in turn, can be eliminated in virtue of Theorem 4.5. In the case of 0-ary connectives, remember that cut-coherence can be proved with the empty theory and then, no further applications of RCUT are introduced. Moreover, the cases when the cut-formula is an (OL) atomic proposition can be easily eliminated.

The non-principal cases are handled as usual: the application of the inference rule is permuted down the cut, thus reducing the height of the cut.

By induction on the size n of the cut-formula and from `OLCutStep` we conclude

```
Theorem OLCutElimination : IsPositiveAtomFormulaL Gamma →
  (( OLTheoryCut n) |-- Gamma ; []  ; (>[]))  → ( OLTheory |-- seq ; Gamma ; []  ;  (>[])  .
```

This means that the theories `(OLTheoryCut n)` and `OLTheory` prove the same sequents and hence, the cut-rule, at the object level, is admissible.

This concludes the proof of cut-elimination for cut-coherent OLs featuring weakening and contraction on both sides of the sequent. Since the rules of LK adhere to this condition and the encoding was proved to be adequate, what we obtain is a proof of cut-elimination for first-order classical logic.

5.4 Controlling the structural rules

The encoding in the previous section marks the atoms $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ with the exponential ?, thus capturing the fact that formulas can be weakened and contracted. What if the structural rules apply only on the left side of the sequent (as in intuitionistic systems) or they do not apply at all (as in substructural logics)? This section addresses this situation by defining an adequate cut-rule for those systems and proving that cut-coherence implies cut-elimination of the OL.

Intuitionistic systems. In single-conclusion systems, the structural rules apply on the left side of the sequent but not on the right side. Consider for instance the system LJ for intuitionistic logic. Sequents are of the shape $\Gamma \vdash F$ where F is a formula. Such a sequent will be encoded as the FLL sequent $\vdash \lfloor\Gamma\rfloor : \lceil F\rceil \Uparrow$ where the right formula is stored into the linear context while the left formulas are stored into the classical context.

Below we encode the rules for implication in LJ:

$$\dfrac{\Gamma \vdash A \quad \Gamma, B \vdash G}{\Gamma, A \to B \vdash G}\to_L \qquad \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\to_R \qquad \begin{array}{l} \to_L\!: \exists A, B.\lfloor A \to B\rfloor^\perp \otimes (!\lceil A\rceil \otimes ?\lfloor B\rfloor) \\ \to_R\!: \exists A, B.\lceil A \to B\rceil^\perp \otimes (?\lfloor A\rfloor \parr \lceil B\rceil) \end{array}$$

The use of ! in the body of $\to_L$ is fundamental to guarantee that the encoding is adequate at the level of derivations: the proof of $!\lceil A\rceil$ must consider only the classical context and then, the atom $\lceil G\rceil$ must necessarily be placed on the left premise of this rule:

$$\dfrac{\dfrac{\dfrac{}{\vdash \lfloor\Gamma, A \to B\rfloor : \cdot \Downarrow \lfloor A \to B\rfloor^\perp}[I_2] \quad \dfrac{\dfrac{\vdash \lfloor\Gamma, A \to B\rfloor : \lceil A\rceil \Uparrow \cdot}{\vdash \lfloor\Gamma, A \to B\rfloor : \cdot \Downarrow !\lceil A\rceil}[!\,R \Downarrow R \Uparrow] \quad \dfrac{\vdash \lfloor\Gamma, A \to B, B\rfloor : \lceil G\rceil \Uparrow \cdot}{\vdash \lfloor\Gamma, A \to B\rfloor : \lceil G\rceil \Downarrow ?\lfloor B\rfloor}[R \Downarrow ?]}{\vdash \lfloor\Gamma, A \to B\rfloor : \lceil G\rceil \Downarrow !\lceil A\rceil \otimes ?\lfloor B\rfloor}[\otimes]}{\vdash \lfloor\Gamma, A \to B\rfloor : \lceil G\rceil \Downarrow \lfloor A \to B\rfloor^\perp \otimes (!\lceil A\rceil \otimes ?\lfloor B\rfloor)}[\otimes]}{\vdash \lfloor\Gamma, A \to B\rfloor : \lceil G\rceil \Uparrow \cdot}[D_3 \exists\exists]$$

Hence, we obtain two premises, each of them corresponding to the premises after the application of the rule at the object level. The other rules of LJ can be encoded similarly and we can prove that the encoding of LJ as the LL theory `OLTheory` is adequate (see `LJ.v`):

```
Theorem Adequacy: isOLFormulaL Gamma → isOLFormula G →
(LJSeq Gamma G)   ↔  seq OLTheory (LEncode Gamma) [ u|G |  ]  (> [])).
```

The cut-rule $\dfrac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash G}\,Cut_i$ is encoded as `Definition RCUTi F := !u|F| ** ?d|F|.` Again, the exponential ! guarantees that the resulting premises will have exactly one atom of the form $\lceil\cdot\rceil$ in the linear context.

Using `RCUTi`, it is possible to prove that the bodies of the encoded rules are cut-coherent. Hence, the principal cases during the cut-elimination procedure can be eliminated by using the cut-rule at the meta-level (FLL) as we explained in the previous section. We thus obtain

```
Theorem OLCutElimination : IsPositiveAtomFormulaL Gamma → IsOLFormula G →
   (( OLTheoryCuti n) |-- Gamma ; [r| G|]   ; (>[]))  → ( OLTheory |-- seq ; Gamma ; [r| G|]  ; (>[])  .
```

where `OLTheoryCuti` corresponds to the theory `OLTheory` extended with `RCUTi`.

Substructural logics. If the OL at hand does not admit structural rules neither on the left- nor on the right-hand side of the sequent, the resulting encoding must reflect this fact by storing the encoded formulas in the linear context. For instance, in a two-sided presentation of MALL, sequents of the form $\Gamma \vdash \Delta$ are encoded as $\vdash \cdot : \lfloor \Gamma \rfloor, \lceil \Delta \rceil \Uparrow \cdot$. For instance, the rules for additive conjunction in MALL are encoded as follows:

$$\frac{\Gamma, A_i \vdash \Delta}{\Gamma, A_1 \& A_2 \vdash \Delta}\ \&_{L_i} \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \& B, \Delta}\ \&_R \qquad \begin{array}{l} \&_L : \exists A, B. \lfloor A \& B \rfloor^{\perp} \otimes (\lfloor A \rfloor \oplus \lfloor B \rfloor) \\ \&_R : \exists A, B. \lceil A \& B \rceil^{\perp} \otimes (\lceil A \rceil \& \lceil B \rceil) \end{array}$$

In `MALL.v` we encode the other rules of the system and prove adequacy:

```
Theorem Adequacy: isOLFormulaL Gamma → isOLFormulaL Delta →
(MALLSeq Gamma Delta)  ↔  seq OLTheory []  (  (LEncode Gamma) ++ (REncode Delta)) (> [])).
```

5.5 Hybrid linear logic (HyLL)

HyLL (Despeyroux and Chaudhuri, 2014) is a conservative extension of intuitionistic LL where the truth judgments are labeled by worlds of the form $F@w$ ("F is true at world w"). In fact, w is an expression built from a given monoid $\langle W, \bullet, \iota \rangle$.

HyLL sequents take the form $\Gamma : \Delta \vdash G@w$ where Γ is the unbounded context (weakening and contraction apply to it) and Δ is the linear context. These sequents will be encoded as the FLL sequent $\vdash \lfloor \Gamma \rfloor : \lfloor \Delta \rfloor, \lceil G@w \rceil \Uparrow .$

The syntax of HyLL includes the LL connectives plus the hybrid connectives:

$F, G ::= p \mid F \otimes G \mid 1 \mid F \multimap G \mid F \& G \mid \top \mid F \oplus G \mid 0 \mid !F \mid \forall x.F \mid \exists x.F \mid (F \ \mathtt{at} \ w) \mid \downarrow w.F \mid \forall w.F \mid \exists w.F$

The last three connectives bind the world w in F. The expression F at w is a mobile proposition: it carries with it the world at which it is true. The formula $(\downarrow u.F)@w$ fixes F to the world w and $\forall w.F$ means that F holds in all world w (similarly for $\exists w.F$).

The specification of this syntax follows as in Section 5.1 but, in $\lfloor X \rfloor$ and $\lceil X \rceil$, we assume that X is a well-formed judgment:

```
Inductive isWorldExp : uexp → Prop :=
  | isWorldExp' : forall w:W, isWorldExp (t_world w)
  | isWorldExp'' : forall wexp wexp', isWorldExp wexp → isWorldExp wexp' → isWorldExp (t_wop wexp wexp')
Inductive isOLJudgment : uexp → Prop :=
  | isOL :  forall F wexp , isOLFormula F → isWorldExp wexp → isOLJudgment (F @ wexp).
```

The set W is a parameter of the specification defining the carrier set of the monoid. Legal world expressions are built from elements in W (`t_world w`) or combining valid expressions through $\bullet$ (`t_wop wexp wexp'`). Hence, $F@w$ is a valid judgment whenever F is a valid formula built from the syntax above and w is a valid world expression.

In the previous section, the exponential ! was used to adequately specify intuitionistic systems. In HyLL, this does not work since the resulting FLL sequents may have several formulas in the linear context: exactly one formula of the shape $\lceil G@v \rceil$ and zero or more (left) atoms $\lfloor F@w \rfloor$. In Chaudhuri et al. (2019a), the HyLL inference rules were encoded as an LL theory and the problem of controlling the right formula $\lceil G@v \rceil$ was solved in the encoding of linear implication:

$$\frac{\Gamma : \Delta, \vdash F@w \quad \Gamma : \Delta', F'@w \vdash G@v}{\Gamma : \Delta, \Delta', F \multimap F'@w \vdash G@v} \multimap_L \quad \frac{\Gamma : \Delta, G@v \vdash G'@v}{\Gamma : \Delta \vdash G \multimap G'@v} \multimap_R$$

$$\multimap_L : \exists F, F', w, G, v. \lfloor F \multimap F' \ @w \rfloor^{\perp} \otimes \lceil G@v \rceil^{\perp} \otimes (\lceil F@w \rceil \otimes (\lfloor F'@w \rfloor \invamp \lceil G@v \rceil))$$

$$\multimap_R : \exists G, G', v. \lceil G \multimap G'@v \rceil^{\perp} \otimes (\lfloor G@v \rfloor \invamp \lceil G'@v \rceil)$$

Note that in $\multimap_L$, after consuming the atom representing the principal formula, the (unique) right formula of the shape $\lceil G@v \rceil$ is also consumed. Later, $\lceil G@v \rceil$ is necessarily stored together with the formula $\lfloor F'@w \rfloor$. This adequately captures the behavior of $\multimap_L$ in HyLL.

The reader can find the complete encoding in Chaudhuri et al. (2019a) and the proof of adequacy in `HyLL.v`:

```
Theorem Adequacy: isOLJudgmentL Gamma → isOLJudgmentL L → isOLJudgment (F @ w) →
 HyLL Gamma L (F @ w) ↔ seq OLTheory (CLEncode Gamma) (REncode (F @ w) :: (LEncode L)) (> []).
```

Due to the careful management of the unique right formula in the encoding, the above adequacy result is at the level of derivations, similar to those presented in the previous sections. This means that every proof in HyLL can be exactly mimicked by a derivation in FLL and hence, HyLL is not more expressive than LL. However, the bodies of $\multimap_L$ and $\multimap_R$ are not cut-coherent and we cannot prove the cut-elimination theorem for HyLL through the cut-coherence argument.

In `HyLLCut.v`, we present an alternative encoding of HyLL without the units ($\top$, 1, 0). This encoding is adequate at the level of proofs. For instance, the rule $\multimap_R$ remains as above and the left rule is encoded as: $\multimap_L : \exists F, F', w. \lfloor F \multimap F'@w \rfloor^{\perp} \otimes (\lceil F@w \rceil \otimes \lfloor F'@w \rfloor)$. Note that, on the sequent $\vdash \lfloor \Gamma \rfloor : \lfloor F \multimap F'@w, \Delta, \Delta' \rfloor, \lceil G@v \rceil \Uparrow$, focusing on $\multimap_L$ may produce as premises the following sequents: $\vdash \lfloor \Gamma \rfloor : \lfloor \Delta, F'@w \rfloor \Uparrow$ and $\vdash \lfloor \Gamma \rfloor : \lfloor \Delta' \rfloor, \lceil G@v \rceil, \lceil F@w \rceil \Uparrow$. In the first (resp. second) sequent, we have only left formulas (resp. two right formulas). This is clearly not the image of any valid HyLL sequent. Fortunately, we can prove the following:

```
Theorem OnlyLeft: ~ seqN OLTheory n (LEncode Gamma) (LEncode L) (> []).
```

In words, there is no proof of a sequent with only left atoms. This is proved by induction on n. We know that the only possibility is to focus on one of the rules in `OLTheory`. The proof cannot start by focusing on the initial rule (since there is no a formula $\lceil G@v \rceil$ in the context). The other cases follow from induction.[3] Relying on the above theorem, we can prove the following:

```
Theorem Adequacy: isOLJudgmentL Gamma → isOLJudgmentL L → isOLJudgment (F @ w) →
 HyLL Gamma L (F @ w) ↔ seq OLTheory (LEncode Gamma) (REncode (F @ w) :: (LEncode L)) (> []).
```

Moreover, we can use the (linear) cut-rule `RCUT1` to prove cut-coherence for this encoding, thus obtaining cut-elimination for HyLL (and hence, also for intuitionistic MALL):

```
Theorem OLCutElimination : IsPositiveAtomFormulaL Gamma →
   (( OLTheoryCutl n) |-- []  ;  Gamma  ;  (>[]))  → ( OLTheory |-- seq ; []  ;  Gamma ; (>[])  .
```

5.6 Sequent calculus and natural deduction

Our last application considers the equivalence of the LL formulas specifying different inference systems (Miller and Pimentel, 2013). We start by encoding the elimination and introduction rules of a natural deduction system for minimal intuitionistic logic (see `NDSeq.v`):

$$\frac{F \to G \quad F}{G}\ \to_E \qquad \frac{\begin{matrix}[F]\\ G\end{matrix}}{F \to G}\ \to_I \qquad \frac{\vdash F_1 \wedge F_2}{\vdash F_i}\ \wedge_{Ei} \qquad \frac{\vdash F \quad \vdash G}{\vdash F \wedge G}\ \wedge_I \qquad \frac{\vdash \forall x.F}{\vdash F[t/x]}\ \forall_E \quad \frac{\vdash F[c/x]}{\vdash \forall x.F}\ \forall_I$$

$$\to_E := \exists F, G.\lceil G\rceil^{\perp} \otimes ! \lceil F\rceil \otimes ! \lceil F \to G\rceil \qquad \to_I := \exists F, G.\lceil F \to G\rceil^{\perp} \otimes (?\lceil F\rceil \invamp \lceil G\rceil)$$

$$\wedge_{Ei} := \exists F_1, F_2.((\lceil F_1\rceil^{\perp} \oplus \lceil F_2\rceil^{\perp}) \otimes ! \lceil F_1 \wedge F_2\rceil) \qquad \wedge_I := \exists F, G.\lceil F \wedge G\rceil^{\perp} \otimes (\lceil F\rceil \,\&\, \lceil G\rceil)$$

$$\forall_E := \exists F, t.(\lceil F\ t\rceil^{\perp} \otimes ! \lceil \forall x.F\rceil) \qquad \forall_I := \exists F.\lceil \forall x.F\rceil^{\perp} \otimes \forall x.\lceil F\ x\rceil$$

These rules faithfully capture the inference rules and adequacy can be proved at the level of derivations. Consider the encoding for conjunction, implication, and the universal quantifier for the system LJ in Section 5.4. Similarly as we did for cut-coherence, we use the specification of the cut-rule to prove the equivalences $\to_E \equiv \to_L$, $\to_I \equiv \to_R$ $\wedge_E \equiv \wedge_L$, $\wedge_I \equiv \wedge_R$, $\forall_E \equiv \forall_L$, and $\forall_I \equiv \forall_R$.

Let `ND` and `SQ` be the theories containing, respectively, the rules for natural deduction and sequent calculus for first-order minimal logic (connectives →, ∧, and ∀) as well as the rule `RCUT`. Using the above equalities, and the FLL cut-rule, we can prove that:

```
Theorem Adequacy: forall F, isOLFormula F →
  (ND |-- [ atom (up F) ] ; (> [ ]))   ↔ (SQ |-- [ atom (up F) ] ; (> [ ]))
```

Therefore, we are using the metatheory of LL (and its cut-elimination theorem) to establish a fundamental result for first-order minimal logic: the mutual relative completeness of natural deduction and sequent calculus.

6. Related and Future Work

The resource awareness of LL has found application in many different areas, including the specification and verification of programming languages, concurrent systems and, as illustrated here, proof systems. The cut-elimination and completeness of focusing theorems are central to all these applications. We have proposed a novel proof of cut-elimination for FLL and, in Xavier et al. (2017), we mechanized the proof of completeness of focusing. Having a formalization of these theorems is of paramount importance because they have served as the foundation to many results in the literature. In particular, we have used our framework to formalize the results in Miller and Pimentel (2013) and Chaudhuri et al. (2019a). We have presented alternative encodings to the ones in Miller and Pimentel (2013) and proposed an encoding of HyLL that satisfies cut-coherence. Other applications such as the encoding of transition systems and the proof of reachability properties can be found in the repository.

When developing our framework, along the way we also improved Hybrid by adding a variety of new lemmas and proof techniques, trying to keep them as general as possible so that they can be applied easily to other OLs and future proof developments. With the support for syntax provided by Hybrid, it was straightforward to consider OLs with binders including first-order quantifiers and worlds' binders in HyLL. Future work on Hybrid will include new infrastructure for increasing the class of properties that can be proved about OLs with quantifiers; this work will include, for example, eliminating the axiom `OLSize` introduced in Section 5.3.

Intuitionistic propositional LL was implemented in Coq (Power and Webster, 1999) and Isabelle (Kalvala and Paiva, 1995), but the main goal in those papers was to provide proof search, and thus no meta-theorems were proved. Cut-elimination and invertibility lemmas were proved for a formalization of several LL calculi in Abella (Chaudhuri et al., 2019b). The reader may also refer to YALLA,[4] and embedding of propositional LL in Coq.

A generic method for formalizing sequent calculi in Isabelle/HOL is proposed in Dawson and Goré (2010). The meta-theorems are parametrized by the set of rules and for cut-elimination, weakening must be admissible. The authors have applied the method to provability logics.

Completeness of focusing was proved for intuitionistic propositional logic in Twelf and Agda (Simmons, 2014). The proof follows the technique developed in Pfenning (2000) where sequents are annotated with terms and the problem is reduced to type-checking. The method in Simmons (2014) could be used to prove completeness of focusing of LL as well. A possible candidate implementation is the one from Reed (2009) on a modified version of Twelf. Unfortunately, this development was never carried out.

When proving the cut-elimination property for focused systems, it is natural to have different cut-rules handling the different phases of the proof. For instance, in Liang and Miller (2009), seven cut-rules were used in the cut-elimination procedure for LJF (a focused proof system for intuitionistic logic). The proof in Abella[5] of cut-elimination for focused propositional MALL uses three different cut-rules. The rules and procedure described in Section 4 seem to be the first cut-elimination procedure for full first-order focused LL.

As illustrated here, LL is general enough for capturing intuitionistic and classical behaviors in OLs. However, in order to specify more complex modalities such as the ones of S4 or substructural features in multi-conclusion intuitionistic systems, LL is not enough and extensions of it are needed, for example, subexponential LL (SELL, see e.g., Nigam et al. (2016, 2017)). Alternatively, in Lellmann et al. (2017), it is shown how a linear nested sequent (LNS) calculus for LL can be used as the basis for encoding, in a modular way, different modalities of OLs. In Olarte et al. (2020), we have shown how to extend the notion of cut-coherence for LNS-based systems, thus allowing us to prove cut-elimination for different modal logics. We are currently working on a extension of our Coq implementation to consider, as SL, a focused LNS presentation of LL and prove the cut-elimination results in *op cit*.

Acknowledgments. The work of Olarte was funded by CNPq. We thank the anonymous reviewers for helping us to improve the material of this paper. We also thank Elaine Pimentel for the lively discussions about the results in Section 5.

Notes

[1] https://coq.inria.fr/refman/proof-engine/ltac

[2] https://github.com/PrincetonUniversity/certicoq

[3] If the units are considered, the above result does not hold since, for example, the HyLL formula $0@w$ on the left finishes any sequent. In that case, we have to analyze also the cases of (ill-formed) sequents with two right formulas and show that, indeed, non-legal applications of the rules cannot prove more OL sequents.

[4] https://perso.ens-lyon.fr/olivier.laurent/yalla/

[5] https://github.com/meta-logic/abella-reasoning

References

Ambler, S., Crole, R. L. and Momigliano, A. (2002). Combining higher order abstract syntax with tactical theorem proving and (co)induction. In: Carreño, V., Muñoz, C. A. and Tahar, S. (eds.), *15th International Conference, TPHOLs 2002*, vol. 2410. *Lecture Notes in Computer Science*. Springer, 13–30.

Andreoli, J.-M. (1992). Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation* **2** (3).

Bertot, Y. and Castéran, P. (2004). *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS *Series*, Springer.

Caires, L., Pfenning, F. and Toninho, B. (2016). Linear logic propositions as session types. *Mathematical Structures in Computer Science* **26** (3) 367–423.

Cervesato, I. and Pfenning, F. (2002). A linear logical framework. *Information & Computation* **179** (1) 19–75.

Chaudhuri, K., Despeyroux, J., Olarte, C. and Pimentel, E. (2019a). Hybrid linear logic, revisited. *Mathematical Structures in Computer Science* **29** (8) 1151–1176.

Chaudhuri, K., Lima, L. and Reis, G. (2019b). Formalized meta-theory of sequent calculi for linear logics. *Theoretical Computer Science* **781** 24–38.

Chlipala, A. (2008). Parametric higher-order abstract syntax for mechanized semantics. In: Hook, J. and Thiemann, P., (eds.), *Proc. of ICFP*, pp. 143–156. ACM.

Crole, R. L. 2011. The representational adequacy of Hybrid. *Mathematical Structures in Computer Science*, **21**(3):585–646.

Dawson, J. E. and Goré, R. (2010). Generic methods for formalising sequent calculi applied to provability logic. In: *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17*.

Despeyroux, J. and Chaudhuri, K. (2014). A hybrid linear logic for constrained transition systems. In: *Post-Proc. of TYPES 2013*, vol. 26. *Leibniz Intl. Procs. in Informatics*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 150–168.

Despeyroux, J., Felty, A. P. and Hirschowitz, A. (1995). Higher-order abstract syntax in Coq. In: *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA*.

Despeyroux, J., Felty, A. P., Liò, P. and Olarte, C. (2018). A logical framework for modelling breast cancer progression. In: Chaves, M. and Martins, M. A. (eds.), *MLCSB 2018*, vol. 11415. *LNCS*. Springer, 121–141.

Felty, A. and Momigliano, A. (2009). Reasoning with hypothetical judgments and open terms in Hybrid. In: *Principles and Practives of Declarative Programming (PPDP)*. ACM, 83–92.

Felty, A. P. and Momigliano, A. (2012). Hybrid - A definitional two-level approach to reasoning with higher-order abstract syntax. *Journal of Automated Reasoning* **48** (1) 43–105.

Gentzen, G. (1969). Investigations into logical deductions. In: *The Collected Papers of Gerhard Gentzen*. North-Holland, 68–131.

Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science* **50** 1–102.

Kalvala, S. and Paiva, V. D. (1995). Mechanizing linear logic in Isabelle. In: *In 10th International Congress of Logic, Philosophy and Methodology of Science*.

Lellmann, B., Olarte, C. and Pimentel, E. (2017). A uniform framework for substructural logics with modalities. In Eiter, T. and Sands, D. (eds.), *LPAR-21*, vol. 46. *EPiC Series in Computing*. EasyChair, 435–455.

Liang, C. and Miller, D. (2009). Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science* **410** (46) 4747–4768.

Mahmoud, M. Y. and Felty, A. P. (2019). Formalization of metatheory of the Quipper quantum programming language in a linear logic. *Journal of Automated Reasoning* **63** (4) 967–1002.

McDowell, R. and Miller, D. (2002). Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic* **3** (1) 80–136.

Miller, D. and Palamidessi, C. (1999). Foundational aspects of syntax. *ACM Computing Surveys* 31.

Miller, D. and Pimentel, E. (2013). A formal framework for specifying sequent calculus proof systems. *Theoretical Computer Science* **474** 98–116.

Miller, D. and Saurin, A. (2007). From proofs to focused proofs: A modular proof of focalization in linear logic. In: Duparc, J. and Henzinger, T. A. (eds.), *CSL*, vol. 4646. *LNCS*. Springer, 405–419.

Nigam, V. and Miller, D. (2010). A framework for proof systems. *Journal of Automated Reasoning* **45** (2) 157–188.

Nigam, V., Olarte, C. and Pimentel, E. (2017). On subexponentials, focusing and modalities in concurrent systems. *Theoretical Computer Science* **693** 35–58.

Nigam, V., Pimentel, E. and Reis, G. (2016). An extended framework for specifying and reasoning about proof systems. *J. Log. Comput.*, **26**(2):539–576.

Olarte, C., Pimentel, E. and Rueda, C. (2018). A concurrent constraint programming interpretation of access permissions. *Theory and Practice of Logic Programming* **18** (2) 252–295.

Olarte, C., Pimentel, E. and Xavier, B. (2020). A fresh view of linear logic as a logical framework. *Electronic Notes in Theoretical Computer Science* **351** 143–165.

Pfenning, F. (2000). Structural cut elimination. *Information and Computation* **157**(1–2) 84–141.

Pfenning, F. and Elliott, C. (1988). Higher-order Abstract Syntax. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

Pimentel, E., Nigam, V. and Neto, J. (2015). Multi-focused proofs with different polarity assignments. In: Benevides, M. R. F. and Thiemann, R. (eds.) *Proceedings of LSFA 2015*, vol. 323. *ENTCS*. Elsevier, 163–179.

Power, J. and Webster, C. (1999). Working with linear logic in COQ. In *12th International Conference on Theorem Proving in Higher Order Logics*, 1–16.

Reed, J. (2009). *A Hybrid Logical Framework*. PhD thesis, Carnegie Mellon University.

Simmons, R. J. (2014). Structural focalization. *ACM Transactions on Computational Logic* **15** (3), 21:1–21:33.

Xavier, B., Olarte, C., Reis, G. and Nigam, V. (2017). Mechanizing focused linear logic in coq. In Alves, S. and Wasserman, R. (eds.), *LSFA 2017*, vol. 338. *ENTCS*. Elsevier, 219–236.

Cite this article: Felty A, Olarte C and Xavier B (2021). A focused linear logical framework and its application to metatheory of object logics. *Mathematical Structures in Computer Science*. https://doi.org/10.1017/S0960129521000323