A Meta Linear Logical Framework

Andrew McCreight and Carsten Schürmann

Yale University
New Haven, CT, USA
{aem|carsten}@cs.yale.edu

Abstract. Over the years, logical framework research has produced various type theories designed primarily for the representation of deductive systems. Reasoning about these representations requires expressive special purpose meta logics, that are in general not part of the logical framework. In this work, we describe $\mathcal{L}_\omega^+$, a meta logic for the linear logical framework LLF [CP96] and illustrate its use via a proof of the admissibility of cut in the sequent calculus for the tensor fragment of linear logic.

1 Introduction

Logical frameworks are meta languages designed for representing various formal systems prevalent in programming language semantics, logics, and protocol design. By definition, logical frameworks are logic independent. This means they are concerned primarily with the way systems are represented, and not necessarily with how to reason about their properties. In this spirit, advanced logical frameworks may incorporate linear types to model resource aware formal systems, such as programming languages with effects, ordered types for systems that access resources in a particular order, and even monadic types to capture concurrency.

The absence of meta logics that facilitate reasoning about encodings in a logical framework is a severe impediment to the deployment and acceptance of the technology among researchers and scientists as well as developers and industry. Consequently, the prevalent use of logical framework technology is as a representation language for one particular logic that is then used to describe and reason about the object systems in question. Higher-order logic is a popular candidate used in Isabelle/HOL [Pau94] and Twelf/HOL [App01] which have been instrumental in the formal study of programming languages, such as Java [NvO98], hardware verification [Har97], and protocol verification [Pau97], and many others. Higher-order logic is well-understood, clean, expressive, and when enriched with induction principles a good choice for many applications. However, it limits the ways in which the respective systems are encoded, and therefore cannot take advantage of the advanced representation technology provided by modern logical frameworks.

This means that a different and possibly better scalable approach to modeling object systems lies in the direct use of a logical framework without taking a

detour through other logics. In this setting properties and proofs are expressed in special purpose meta logics, tailored to a particular underlying logical framework, designed solely for the purpose of reasoning and not modeling. A logical framework together with a meta logic defines a meta logical framework. $\mathcal{M}_\omega^+$ [Sch00], for example, is a meta logic designed for the logical framework LF [HHP93].

In this work, we describe a special purpose meta logic for the linear logical framework LLF [CP96], resulting in a meta linear logical framework. LLF's distinguishing feature over LF is a set of linear operators capable of handling depletable resources. LLF has been successfully employed in representing and experimenting with a variety of security and authentication protocols [CDL$^+$99]. Although the theory behind LLF is well-understood, this is to our knowledge the first research towards a sound meta logic for LLF that is also amenable to automated proof search.

The soundness of the meta linear logical framework is based on a realizability argument. We develop a meta logic together with a system of proof terms that compute effectively with LLF objects and argue that each proof term corresponds to a total function. Though the two necessary syntactic criteria to ensure termination and coverage are not described in this paper, we stipulate that they exist.

The quest for meta logics for various logical frameworks is not new. The $FO\lambda^{\Delta\mathbb{N}}$ system [MM97], for example, relies on a sequent calculus for first-order logic extended by definition and natural number induction and supports the encoding of frameworks based on hereditary Harrop formulas, and even linear logic [McD97]. In $FO\lambda^{\Delta\mathbb{N}}$, however, induction is limited to natural numbers, whereas $\mathcal{L}_\omega^+$ supports induction over canonical derivations in LLF (that are inductively defined). In this respect it differs also from a meta-logical framework based on rewriting logic [BCM00] because it lacks initial models. This work extends the methodology that led to $\mathcal{M}_\omega^+$ [Sch00] to the linear case.

The paper is organized in the following way: in Section 2 we review the linear logical framework LLF and illustrate its representational expressiveness in terms of a sequent calculus for the tensor fragment of linear logic. Next, in Section 3, we explain informally what it means to reason *about* derivations. We use as an example the proof of the theorem that cuts are admissible. In Section 4, we present a formal meta-logic $\mathcal{L}_\omega^+$ that serves as the formalization of theorems as well as meta-theoretic proofs. $\mathcal{L}_\omega^+$'s soundness is shown in Section 5 before we conclude in Section 6 and assess results.

2 The Linear Logical Framework LLF

A logical framework [Pfe99] is a meta language suitable for representing deductive systems that are commonplace in programming language, logic, and type theory research, and are used to describe operational semantics, type systems, proof theories, and program transformers.

The logical framework LF [HHP93], for example, provides dependent types, a conceptually weak function space, and canonical forms, which render it an el-

$$\frac{}{A \Longrightarrow A}\,\mathsf{ax} \qquad \frac{\Gamma_1 \Longrightarrow C, \Delta_1 \qquad \Gamma_2, C \Longrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}\,\mathsf{cut}$$

$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \otimes B \Longrightarrow \Delta}\otimes\mathsf{L} \qquad \frac{\Gamma_1 \Longrightarrow A, \Delta_1 \qquad \Gamma_2 \Longrightarrow B, \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow A \otimes B, \Delta_1, \Delta_2}\otimes\mathsf{R}$$

Fig. 1. Tensor fragment of linear logic

egant tool for adequate encodings of judgments as types, derivations as objects, and hypothetical judgments as functions. For example, the well-known derivability judgment for classical logic of the form $A_1, \ldots, A_n \Longrightarrow B_1, \ldots, B_n$ can be represented in LF as a function of the form

$$\text{neg } A_1 \ldots \to \text{neg } A_n \to \text{pos } B_1 \ldots \to \text{pos } B_m \to \#.$$

neg and pos are families of types, representing assumptions to the left and right of the $\vdash$ symbol, respectively, while # is a type that stands for the empty sequent. Encoding lists of assumptions this way instead of making them explicit as lists has several advantages, namely that lookup and substitution are directly supported by LF through variables names and β-reduction, which renders the formal description of inference brief, concise, and readable.

One limitation of LF is that it confines encodings to the intuitionistic properties of the LF function space. In classical logic, no assumption can be retracted within a derivation. But this is exactly what one might want for other applications, such as programming with state [CP96], or the analysis of security protocols [CDL$^+$99].

The linear logical framework LLF [CP96] therefore extends the logical framework LF [HHP93] by resource handling constructs from linear logic [Gir87] with $\beta\eta$ as definitional equality [Coq91]. It distinguishes binding constructs for intuitionistic assumptions from those of linear ones. The linear function space is denoted with $A \multimap B$. There is no dependent linear function space. As an example, consider the representation of the tensor fragment of classical linear logic depicted in Figure 1. The rules cut and $\otimes\mathsf{R}$ illustrate how resources on either side of the sequent symbol are distributed as resources to either of the two premisses. A derivation can only then be closed by ax if the left and the right context contain a single formula A. A sequence $A_1, \ldots, A_n \Longrightarrow B_1, \ldots, B_n$ in linear logic is therefore represented as an object of type

$$\text{neg } A_1 \multimap \ldots \multimap \text{neg } A_n \multimap \text{pos } B_1 \multimap \ldots \multimap \text{pos } B_m \multimap \#,$$

where each of the inference rules is represented as a constant in LLF as shown in Figure 2. As usual, we omit the leading Π-quantifiers for inferable types. LLF's meta theory guarantees the existence of β-normal, η-long canonical forms [VC02] used in order to establish the adequacy of this encoding.

$$\begin{array}{ll}
\text{ax} : \text{neg } A \multimap \text{pos } A \multimap \#. & \text{cut} : (\text{pos } C \multimap \#) \multimap (\text{neg } C \multimap \#) \multimap \#. \\
\text{tensorR} : (\text{pos } A \multimap \#) \multimap (\text{pos } B \multimap \#) \multimap (\text{pos } (A \otimes B) \multimap \#). & \text{tensorL} : (\text{neg } A \multimap \text{neg } B \multimap \#) \multimap (\text{neg } (A \otimes B) \multimap \#).
\end{array}$$

Fig. 2. Encoding of Figure 1 in LLF

3 Meta Linear Logical Reasoning

All of linear logic, including the exponential quantifiers, enjoy the cut-elimination property. This is relatively easy to prove by hand, aside from the multitude of cases to be considered, but it is difficult to carry out such an argument in a formal setting. If we still want to take advantage of the elegant encoding, what kind of formal setting shall we use? What kind of induction principles are valid, which are preferable, and how do they interact with linearity? What kind of logic is most appropriate: first-order, higher-order, temporal, modal?

In the absence of a good answer, these questions have prompted us to develop the meta logic $\mathcal{L}^+_\omega$ for the linear logical framework LLF. $\mathcal{L}^+_\omega$ extends the meta-logic $\mathcal{M}^+_\omega$ for LF developed by the second author [Sch00] into the LLF setting. $\mathcal{L}^+_\omega$ is first-order, intuitionistic, and not linear. Aside from $\top$, it does not define any logical constant symbols. It does however inherit proofs by induction over arbitrary higher-order types without the restrictive positivity condition, including those that take advantage of both linear and intuitionistic assumptions. Furthermore, it supports quantification over LLF contexts.

To illustrate the logic, we consider the proof that cuts are admissible in the tensor fragment of linear logic depicted in Figure 1.

Theorem 1 (Admissibility of cut). *If* $\mathcal{P} :: \Gamma_1 \Longrightarrow C, \Delta_1$ *and* $\mathcal{Q} :: \Gamma_2, C \Longrightarrow \Delta_2$ *then* $\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2$.

Proof. By lexicographic structural induction on the subformula A and simultaneously on $\mathcal{P}$ and $\mathcal{Q}$ [Pfe94]. We show only the essential case between $\otimes\mathsf{R}$ and $\otimes\mathsf{L}$. The remaining cases can be found in Appendix A.

$$\begin{array}{ll}
\mathcal{P} :: \Gamma_1', \Gamma_1'' \Longrightarrow A \otimes B, \Delta_1', \Delta_1'' & \text{(by assumption)} \\
\mathcal{P}_1 :: \Gamma_1' \Longrightarrow A, \Delta_1' & \text{(by assumption)} \\
\mathcal{P}_2 :: \Gamma_1'' \Longrightarrow B, \Delta_1'' & \text{(by assumption)} \\
\mathcal{Q} :: \Gamma_2, A \otimes B \Longrightarrow \Delta_2 & \text{(by assumption)} \\
\mathcal{Q}_1 :: \Gamma_2, A, B \Longrightarrow \Delta_2 & \text{(by assumption)} \\
\mathcal{R}_1 :: \Gamma_1', \Gamma_2, B \Longrightarrow \Delta_1', \Delta_2 & \text{(by ind. hyp. on } \mathcal{P}_1, \mathcal{Q}_1) \\
\mathcal{R} :: \Gamma_1', \Gamma_1'', \Gamma_2 \Longrightarrow \Delta_1', \Delta_1'', \Delta_2 & \text{(by ind. hyp. on } \mathcal{P}_2, \mathcal{R}_1)
\end{array}$$

Formulas, Contexts, and Worlds. The guiding design principle of our system is to focus meta-theoretic reasoning around canonical form derivations and well-formedness derivations of contexts, and not simply terms and contexts. In LF

and LLF alike, every term has a canonical form that can be inductively defined. Our first-order quantifiers range therefore over canonical forms $M : (\gamma \triangleright A)$ and context variables $\gamma \in (\gamma' \triangleright \Phi)$. For the sake of naming, γ must be valid in γ' and of type Φ. Here's the formalization of Theorem 1:

$$\begin{array}{l} \forall \gamma_1 \in (\cdot \triangleright \Phi).\, \forall \gamma_2 \in (\gamma_1 \triangleright \Phi).\, \forall C : (\cdot \triangleright \mathrm{o}). \\ \quad \forall P : (\gamma_1 \triangleright \mathrm{pos}\ C \to \#).\, \forall Q : (\gamma_2 \triangleright \mathrm{neg}\ C \to \#). \\ \quad \exists R : (\gamma_1, \gamma_2 \triangleright \#).\, \top \end{array} \tag{1}$$

where

$$\Phi = ((\lambda A : \mathrm{o}.\, \exists n : \mathrm{neg}\ A.\, \epsilon) + (\lambda A : \mathrm{o}.\, \exists p : \mathrm{pos}\ A.\, \epsilon))^*. \tag{2}$$

The first two quantifiers in (1) range over contexts γ_1 (valid in the empty context $\cdot$) and γ_2 (valid in γ_1). γ_1 represents simultaneously the list of hypotheses Γ_1 and Δ_1, while γ_2 represents Γ_2 and Δ_2. Φ is the type (or *world*) of these contexts, which ensures that γ_1 and γ_2 only contain assumptions of the form "pos A" and "neg A". Worlds are regular expressions that force a regular structure on contexts and organize them in blocks as exhibited by (2). "+" denotes alternative, "*" repetition, and each block starting with $\exists$ is parameterized by formula A : o. The ϵ marks the end of the block. For example,

$$p_1 : \mathrm{pos}\ A_1, p_2 : \mathrm{pos}\ A_2, n_3 : \mathrm{neg}\ A_3 \in \Phi.$$

Worlds have been extensively studied in prior work by the second author [Sch01]. In (1), C ranges over closed formulas, P over sequent derivations in γ_1 with formula C on the left, and Q over sequent derivations in γ_2 with formula C on the right. R stands for the result derivation, necessarily valid in the union of γ_1 and γ_2.

Proofs. Proofs in $\mathcal{L}_\omega^+$ are derivations in a sequent style system. However, to best understand the intricacies of the logical system, we present proofs as functions that act as realizers, comparable to a Curry-Howard isomorphism style argument. In our example, Formula (1) is interpreted as a function type and its proof as a total function that maps contexts Δ_1, Δ_2, and LLF derivations $\cdot; \cdot \vdash C : \mathrm{o}$, $\cdot; \Delta_1 \vdash P : \mathrm{pos}\ C \to \#$, and $\cdot; \Delta_2 \vdash Q : \mathrm{neg}\ C \to \#$ into a derivation $\cdot; \Delta_1, \Delta_2 \vdash R : \#$.

This particular realizer is best presented by cases, closely following the outline of the proof of Theorem 1. Again we only discuss the essential case depicted in Figure 3. The other cases are given in the Appendix B. As for syntactic sugar, we stay as closely as possible to that of a functional programming language, omitting details that do not enhance the reader's understanding. Figure 3 illustrates the novel and distinct features of $\mathcal{L}_\omega^+$ including pattern-matching against linear patterns, hypothetical reasoning, and context splitting . The remainder of this section is structured according to the two most important features "ca" has to satisfy to qualify as a proof: coverage and termination.

```
fun ca (γ'_1, γ''_1) γ_2 (A ⊗ B)
      (λ̂p:pos (A ⊗ B). tensorR^(γ'_1 ▷ P_1)^(γ''_1 ▷ P_2)^p)
      (λ̂n:neg (A ⊗ B).
          tensorL^(γ_2 ▷ (λ̂n_1:neg A. λ̂n_2:neg B. Q_1^n_1^n_2))^n) =
    new
      α :: (γ'_1, γ_2 ▷ ∃n : neg B. ϵ)
    in
      let
        val ⟨R_1, ⟨⟩⟩ = ca γ'_1 (γ_2, α : ∃n : neg B. ϵ) A
                              P_1
                              (λ̂n_1:neg A. Q_1^n_1^π_t(α))]
        val ⟨R, ⟨⟩⟩ = ca γ''_1 (γ'_1, γ_2) B
                             P_2
                             (λ̂n:neg B. R_1[n/π_t(α)])
      in
        ⟨R, ⟨⟩⟩
      end
    end
```

Fig. 3. Admissibility of cut, essential case

Coverage. Contrary to common practice, quantifiers and functions range over canonical form derivations of terms, which has several advantages. First, context and type information are directly accessible, second, canonical forms are inductively defined and can therefore be interpreted as patterns, and third, they form equivalence classes among all β-normal η-long equivalent LLF terms.

Consequently, contexts are resources that can be passed around, matched against, split, and combined. We write γ for variables that range over contexts. The first two arguments to "ca" are the context patterns (γ'_1, γ''_1) and γ_2. How the context is split into γ'_1 and γ''_1 is determined by how the two contexts are being used. This is fixed by ascribing context information to the two variables P_1 and P_2 bound in the fourth argument to "ca": $(\hat{\lambda}p{:}\mathrm{pos}\ (A \otimes B).\, \mathrm{tensorR}\hat{}(\gamma'_1 \triangleright P_1)\hat{}(\gamma''_1 \triangleright P_2)\hat{}p)$. Context and type ascription are features of the syntax we have chosen to present proofs in $\mathcal{L}^+_\omega$, with counterparts in the formal development of $\mathcal{L}^+_\omega$ in Section 4.

The challenge is to verify that "ca" covers all cases. Canonical forms are patterns and in the interest of completeness, two additional cases (that are described in Appendix B) related to "tensorR" must be considered, depending on if p is consumed in P_1 or P_2.

Termination. "ca" must be total in order to be considered a proof. Therefore any evaluation of "ca", independent of what arguments applied, must terminate. Consider the body of "ca" in Figure 3. The two recursive calls to "ca" correspond to appeals of the induction hypothesis in the proof of Theorem 1, yielding result objects R_1 and R, respectively.

The first instruction is the **new** instruction that introduces a new hypotheses of type neg B. Recall from the proof of Theorem 1, that $\mathcal{R}_1$ is a result of the induction hypothesis applied to $\mathcal{P}_1$ and $\mathcal{Q}_1$, which is parametric in B. Since hypothetical arguments are encoded via higher-order functions, "ca" can only execute a recursive call after traversing the binder $(\hat{\lambda} n_2 : \text{neg } B)$. In general one can only do this by applying it to a new parameter $n_2 : \text{neg } B$, in form of the (so called module) declaration

$$\alpha :: (\gamma_1', \gamma_2 \triangleright \exists n : \text{neg } B.\, \epsilon). \tag{3}$$

α is a new variable, that ranges over blocks of new parameters, and is similar to $\underline{x}$ in [Sch01]. Intuitively, one can think of a block as a temporary list of new constant symbols that act as placeholders within the body of **new**. The γ_1', γ_2 resolve all ambiguities related to the naming of α. We write π_{t} to project the head of the list, and π_{m} for the tail. $\pi_{\mathrm{t}}(\alpha)$, for example, is a new name for the newly introduced parameter, and should be used instead of n_2.

The first recursive call cuts P_1 and Q_1 with cut-formula A. Eventually, the computation will finish and the resulting derivation R_1 will use all resources of the set $\gamma_1', \gamma_2, \alpha : \exists n : \text{neg } B.\, \epsilon$, which corresponds directly to the informal proof. Recall that γ_1' represents assumption lists Γ_1' and Δ_1', γ_2 the assumption lists Γ_2 and Δ_2, and α to the additional hypothesis B that occurs to the left of the sequence arrow.

The other recursive call for cutting P_2 and R_1 is similar to the first except that this time the cut formula is B. R_1 is parametric in $\pi_{\mathrm{t}}(\alpha)$, which is subsequently replaced by a linear variable n *before* the second recursive call is invoked. Replacements of this kind are supported in $\mathcal{L}_\omega^+$, expressed by substituting n for $\pi_{\mathrm{t}}(\alpha)$. The resulting R is valid in $\gamma_1', \gamma_1'', \gamma_2$, and does therefore not depend on α. Hence, it can safely escape the scope of **new**.

"ca" terminates because the arguments that correspond to derivations $\mathcal{P}$ and $\mathcal{Q}$ are smaller with respect to a well-founded lexicographical order on the cut formula and simultaneously on $\mathcal{P}$ and $\mathcal{Q}$. In this work, we consider only lexicographic and simultaneous extensions of the subterm ordering. In particular the first recursive call terminates because A and B are subterms of $A \otimes B$.

4 The Meta Logic $\mathcal{L}_\omega^+$

We begin now with the formal description of our proposed meta logic $\mathcal{L}_\omega^+$. The guiding design principle that we have chosen to follow prescribes a rigorous distinction between two levels. The linear logical framework LLF forms one level and includes an advanced type system with a linear function type constructor, additive pairs, and top. LLF serves purely as a representation language for objects that we plan to reason about in $\mathcal{L}_\omega^+$, such as, for example, the sequent calculus for the tensor fragment of linear logic, as presented in Section 2. The meta logic $\mathcal{L}_\omega^+$ defines the other level. It provides the syntactic and proof-theoretic means to express properties about encodings in LLF and their respective proofs if they should exist.

$$
\begin{array}{lll}
(\mathit{Kinds}) & K & ::= \mathrm{type} \mid \Pi u : A.\ K \\
(\mathit{Types}) & A, B & ::= a \mid A\ M \mid \Pi u : A.\ B \mid A \multimap B \mid A\ \&\ B \mid \top \\
(\mathit{Objects}) & M, N & ::= n[\rho] \mid c \mid b \mid \lambda u : A.\ M \mid M\ N \mid \hat{\lambda} u : A.\ M \mid M\ \hat{}\ N \\
 & & \quad \mid \langle M, N \rangle \mid \pi_1 M \mid \pi_2 M \mid \langle\rangle \\
(\mathit{Parameters}) & b & ::= u \mid \pi_{\mathrm{t}} m \\
(\mathit{Signatures}) & \Sigma & ::= \cdot \mid \Sigma, a : K \mid \Sigma, c : A \\
(\mathit{Contexts}) & \Gamma, \Delta & ::= \cdot \mid \Gamma, \gamma \in \Phi \mid \Gamma, b : A \\
(\mathit{Context\ Subst.}) & \rho & ::= \cdot \mid \rho, \gamma/\gamma \mid \rho, M/b
\end{array}
$$

Fig. 4. LLF (variant) syntax

Following the general philosophy underlying this and other meta logical frameworks [Sch00,BCM00] the clear distinction between a language of representation and a language of proofs is essential. Both levels are layered hierarchically which means that the meta logic can access encodings in LLF, construct, deconstruct, compute canonical forms, and appeal to the structural properties that are part of LLF. LLF on the other side does not have any access to theorems and proofs, a restriction which eventually permits us to prove the soundness of $\mathcal{L}_\omega^+$.

4.1 Syntactic Categories

The linear logical framework LLF. Our version of LLF that poses as foundation of $\mathcal{L}_\omega^+$ deviates slightly from the standard formulation of LLF [CP96]. Nothing has changed with respect to layering LLF into objects, types, and kinds. In fact, the level of types, and kinds are completely untouched, the only changes are on the object level.

The parametric nature of canonical derivations as seen from $\mathcal{L}_\omega^+$ requires us to distinguish between local variables that are introduced via abstraction inside a term and variables that stand for an object whose existence is guaranteed by some meta-logical property. For example, an $\mathcal{L}_\omega^+$ theorem may assert the existence of derivations P or Q in (1) that ought to be accessible from within LLF, just as p that is locally bound. Variables of this kind are quite common, represent in fact canonical form derivations, and are denoted with n. To fit n into LLF, it must occur in form of closure under substitution ρ. ρ serves as explicit substitution that is applied to an instantiation of n only during run time. Our extension of LLF with meta-variables occurring under separate contexts with explicit substitutions is similar to a system developed for a different purpose [PP03] (from which we borrow the syntax for the meta-variable binders). Other parameters, such as projections $\pi_{\mathrm{t}}(\alpha)$ from module variables α (see (3)), are also visible to LLF, and are in fact treated no differently than u.

Recall from the function from Figure 3, that LLF derivations are also parametric in terms of context variables γ that can occur within the intuitionistic

$$
\begin{array}{lll}
(\mathit{Mod.\ Kinds}) & k & ::= \mathrm{sig} \mid \Pi u : A.\ k \\
(\mathit{Mod.\ Sigs}) & s & ::= \epsilon \mid \exists u : A.\ s \mid \lambda u : A.s \\
(\mathit{Modules}) & m & ::= \alpha \mid \pi_{\mathrm{m}} m \\
(\mathit{Worlds}) & \Phi & ::= s \mid \Phi^* \mid \Phi_1 + \Phi_2 \\
(\mathit{Environments}) & \chi & ::= \cdot \mid \chi, \gamma \in \Phi \mid \chi, \alpha : s \\
(\mathit{Formulas}) & F & ::= \forall\ n :: (\chi \triangleright A).\ F \mid \forall \gamma \in (\chi \triangleright \Phi).\ F \mid \exists\ n :: (\chi \triangleright A).\ F \mid \top \\
(\mathit{Programs}) & P & ::= \Lambda\ n :: (\chi \triangleright A).\ P \mid \Lambda \gamma \in (\chi \triangleright \Phi).\ P \mid P\ M \mid P\ \chi \\
 & & \quad \mid \langle\!\langle \chi \triangleright M; P \rangle\!\rangle \mid \langle\!\langle\rangle\!\rangle \mid \mathrm{case}\ \Omega \mid x \mid \mu\ x \in F.\ P \mid \nu\ \alpha :: (\chi \triangleright s).\ P \\
(\mathit{Cases}) & \Omega & ::= \cdot \mid \Omega, (\Psi \vdash \sigma \mapsto P) \\
(\mathit{Contexts}) & \Psi & ::= \cdot \mid \Psi, n :: (\chi \triangleright A) \mid \Psi, \gamma \in (\chi \triangleright \Phi) \mid \Psi, x \in F \mid \Psi, \alpha :: (\chi \triangleright s) \\
(\mathit{Substitutions}) & \sigma & ::= \cdot \mid \sigma, M/n \mid \sigma, \chi/\gamma \mid \sigma, P/x \mid \sigma, m/\alpha
\end{array}
$$

Fig. 5. $\mathcal{L}_\omega^+$ syntax

context Γ or the linear context Δ. Those variables represent in general non-empty context extensions, that must be carefully handled by LLF, especially in the axiom case that requires the linear context to contain exactly one hypothesis. (see Appendix C for the rules defining our version of LLF).

In summary, the main differences between our and the standard formulation of LLF are explicit substitutions, context variables, and a mechanism to make hypothetical derivations available to LLF. These additions do not change the fundamental properties of LLF because during runtime each n, α and γ is instantiated by an appropriate LLF variables and LLF contexts, respectively.

The meta logic $\mathcal{L}_\omega^+$. Figure 5 describes the syntactic categories for the meta logic $\mathcal{L}_\omega^+$. Worlds, made out of module signatures, are abstract descriptions of environments (see, for example, (2)). They assert a structure on the context that will be encountered during runtime, but can be exploited within proofs [Sch01] as illustrated by cases 2 and 3 in the definition of "ca" in Appendix A.

Each module has a module kind. If the body of a module does not contain any free variables, the module is said to be of kind sig. If it does, the variables should be properly abstracted, and the module is then of dependent module kind. The world Φ from (2), for example, is defined in terms of $(\lambda A : \mathrm{o}.\, \exists n : \mathrm{neg}\ A.\, \epsilon)$, which is of module kind $(\Pi A : \mathrm{o}.\, \mathrm{sig})$.

$\mathcal{L}_\omega^+$ is a first-order meta-logic that is custom designed for LLF. Similar to $\mathcal{M}_\omega^+$ [Sch00] its syntactic categories consist of formulas, programs, and cases. The most notable change however is that quantifiers range over canonical derivations in LLF valid in individual contexts χ that may be split, passed to the induction hypothesis, and eventually joined with other contexts.

For various reasons, foremost the soundness proof of $\mathcal{L}_\omega^+$, which we discuss in Section 5, each inference rule of $\mathcal{L}_\omega^+$ is endowed with a proof term, a so called program. The first two programs defined in Figure 5 are functions ranging over canonical LLF derivations and contexts, respectively. The next two programs

are applications of a canonical form M and a valid context χ. $\langle\!\langle \chi \triangleright M; P \rangle\!\rangle$ is a proof term for an existential formula, pairing a canonical derivation of an LLF term with a program. Next, we have unit, a case construct with cases Ω, a program variable x, a recursion operator, and finally a new operator. Case and recursion are necessary to represent inductive proofs over the canonical forms in LLF. The formulation of cases $(\Omega, (\Psi \vdash \sigma \mapsto P))$ can be explained as follows. The substitution σ is a pattern, with free variables in Ψ, while P is a program that may contain free variables declared in Ψ. We will explain the form of case programs when we discuss the proof theory of $\mathcal{L}_\omega^+$ in Section 4.4.

The function depicted in Figure 3 is actually a program presented using a lot of syntactic sugar. Without the syntactic sugar, it would be of the form

$$\begin{array}{l} \Lambda\gamma_1 :: (\cdot \triangleright \Phi).\, \Lambda\gamma_2 :: (\gamma_1 \triangleright \Phi).\, \Lambda C :: (\cdot \triangleright \mathrm{o}). \\ \quad \Lambda P :: (\gamma_1 \triangleright \mathrm{pos}\ C \multimap \#).\, \Lambda P :: (\gamma_1 \triangleright \mathrm{neg}\ C \multimap \#). \\ \quad\quad \mathrm{case}\ \Omega \end{array}$$

with $(\Omega, (\Psi \vdash \sigma \mapsto P)) \in \Omega$ where Ψ, σ, and P are defined as follows:

$$\begin{array}{rl} \Psi & = \gamma_1' :: (\cdot \triangleright \Phi), \gamma_1'' :: (\gamma_1' \triangleright \Phi), \gamma_2 :: (\gamma_1', \gamma_1'' \triangleright \Phi), A :: (\cdot \triangleright \mathrm{o}), B :: (\cdot \triangleright \mathrm{o}), \\ & \quad P_1 :: (\gamma_1' \triangleright \mathrm{pos}\ A \multimap \#), P_2 :: (\gamma_1'' \triangleright \mathrm{pos}\ B \multimap \#), \\ & \quad Q_1 :: (\gamma_2 \triangleright \mathrm{neg}\ A \multimap \mathrm{neg}\ B \multimap \#) \\ \sigma & = (\gamma_1', \gamma_1'')/\gamma_1, \gamma_2/\gamma_c, (A \otimes B)/C \\ & \quad (\hat{\lambda} p\!:\!\mathrm{pos}\ (A \otimes B).\, \mathrm{tensorR}\hat{}(\gamma_1' \triangleright P_1)\hat{}(\gamma_1'' \triangleright P_2)\hat{}p)/P, \\ & \quad (\hat{\lambda} n\!:\!\mathrm{neg}\ (A \otimes B).\, \mathrm{tensorL}\hat{}(\gamma_2 \triangleright (\hat{\lambda} n_1\!:\!\mathrm{neg}\ A.\, \hat{\lambda} n_2\!:\!\mathrm{neg}\ B.\, Q_1\hat{}n_1\hat{}n_2))\hat{}n)/Q \\ P & = \nu(\alpha :: (\gamma_1', \gamma_2 \triangleright \exists n : \mathrm{neg}\ B.\, \epsilon)).P' \end{array}$$

P' is the program that corresponds to the remaining **let** construct, expressed as two nested case statements, which we omit in the interest of space.

4.2 Splitting

One of the contributions of this paper is the idea of splitting a context χ that is associated with the canonical derivations bound on the meta-level into the pair of contexts $\Gamma; \Delta$ required by LLF. Some of the declarations in χ will be sorted into Γ and treated intuitionistically, some will end up in Δ and treated linearly, and some may not occur in either. The filtering of the declarations is controlled by a binary predicate P on LLF types. Γ and Δ are created using different predicates. This pair of predicates must guarantee the minimal requirement that the resulting $\Gamma; \Delta$ is a valid LLF context.

Splitting is defined in two stages. First the block structure of χ is separated into individual parameters via *flattening* (see the definition of $[\![m : s]\!]$ and $[\![\chi]\!]$ in Figure 6). Second, the relevant declarations for either context Γ and Δ are selected via the *narrowing* operation ($\lfloor s \rfloor_A^P$, $\lfloor w \rfloor_A^P$, $\lfloor \Phi \rfloor_A^P$ and $\lfloor \Gamma \rfloor_A^P$). We write $[\![\chi]\!]_A^P$ for narrowing composed with flattening ($\lfloor [\![\chi]\!] \rfloor_A^P$) when defining the proof theory for $\mathcal{L}_\omega^+$ in Section 5.

A good choice for each P is one based on the subordination relation [Vir99]. It flags all declarations that may occur in the type of another declaration as

$$
\begin{array}{lll}
\llbracket m : \epsilon \rrbracket & = \cdot & \\
\llbracket m : \exists u : A.\ s \rrbracket & = \pi_{\mathrm{t}} m : A, \llbracket \pi_{\mathrm{m}} m : [\pi_{\mathrm{t}} m/u]s \rrbracket & \\
\lfloor \epsilon \rfloor_A^P & = \epsilon & \\
\lfloor \exists u : B.\ s \rfloor_A^P & = \exists u : B.\ \lfloor s \rfloor_A^P \ \text{ if } P(B,A) & \\
\lfloor \exists u : B.\ s \rfloor_A^P & = \lfloor s \rfloor_A^P \ \text{ if not } P(B,A) & \\
\lfloor \lambda u : A.w \rfloor_A^P & = \lambda u : A.\lfloor w \rfloor_A^P & \\
\lfloor \Phi^* \rfloor_A^P & = (\lfloor \Phi \rfloor_A^P)^* & \\
\lfloor \Phi_1 + \Phi_2 \rfloor_A^P & = \lfloor \Phi_1 \rfloor_A^P + \lfloor \Phi_2 \rfloor_A^P &
\end{array}
\qquad
\begin{array}{lll}
\llbracket \cdot \rrbracket & = \cdot & \\
\llbracket \chi, \gamma \in \Phi \rrbracket & = \llbracket \chi \rrbracket, \gamma \in \Phi & \\
\llbracket \chi, \alpha : s \rrbracket & = \llbracket \chi \rrbracket, \llbracket \alpha : s \rrbracket & \\
\lfloor \cdot \rfloor_A^P & = \cdot & \\
\lfloor \Gamma, \gamma \in \Phi \rfloor_A^P & = \lfloor \Gamma \rfloor_A^P, \gamma \in \lfloor \Phi \rfloor_A^P & \\
\lfloor \Gamma, b : B \rfloor_A^P & = \lfloor \Gamma \rfloor_A^P, b : B \ \text{ if } P(B,A) & \\
\lfloor \Gamma, b : B \rfloor_A^P & = \lfloor \Gamma \rfloor_A^P \ \text{ if not } P(B,A) &
\end{array}
$$

Fig. 6. Flattening and narrowing modulo P

intuitionistic assumptions, and all others (that may actually occur in the object) as linear ones. In all our examples regarding LLF, this choice of predicates was sufficient. If the P for the intuitionistic environment holds for all pairs of LLF types, and the P for the linear environment holds for none, $\mathcal{L}_\omega^+$ reduces to a meta-logic of the logical framework LF [HHP93].

4.3 Semantics of $\mathcal{L}_\omega^+$

The semantic entailment for $\mathcal{L}_\omega^+$ is written in terms of $\models$, a relation that is defined in this section. The main challenge of this relation is to define the meaning of a formula whose quantifiers do not simply range over contexts or terms, but over the respective derivations of canonicity.

$$
\begin{array}{lll}
\models \forall \gamma \in (\chi \triangleright \Phi).\ F & \text{iff} & \models [\chi'/\gamma]F \text{ for all } \cdot\,; \chi \vdash \chi' : \Phi \\
\models \forall\ n :: (\chi \triangleright A).\ F & \text{iff} & \models [M/x]F \text{ for all } \cdot; \llbracket \chi \rrbracket_A^{\prec:}; \llbracket \chi \rrbracket_A^{\hat{\succ}} \triangleright M : A \\
\models \exists\ n :: (\chi \triangleright A).\ F & \text{iff} & \models [M/x]F \text{ for some } \cdot; \llbracket \chi \rrbracket_A^{\prec:}; \llbracket \chi \rrbracket_A^{\hat{\succ}} \triangleright M : A \\
\models \top & &
\end{array}
$$

Variable capture is a problem in a meta logic of this generality, which is solved in our system by requiring that context variables can only be instantiated by contexts that are well-formed in χ. Also, the treatment of linear resources is quite complex. A universal quantifier ranging over LLF objects actually ranges over all canonical form derivations of that object valid in explicit intuitionistic and linear contexts that can be derived from χ via the standard transformation, described in Figure 4.2. The existential is the dual to the universal quantifier, and true is always valid. In the remainder of the paper we give a proof theoretic explanation of this semantics and argue for the soundness of the system in Section 5.

4.4 Proof theory for $\mathcal{L}_\omega^+$

$\mathcal{L}_\omega^+$ is designed as a logic, but the soundness proof to be presented in Section 5 requires it to change its face and become a type theory of total functions. This type theory is defined in Figure 7 for $\mathcal{L}_\omega^+$'s two typing judgments $\Psi \vdash P \in F$

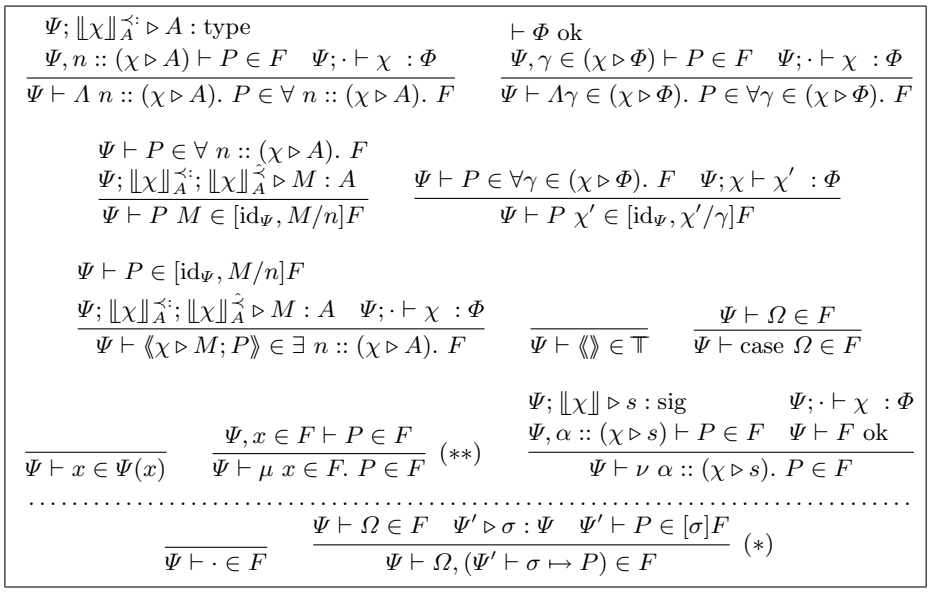

Fig. 7. Derivability in $\mathcal{L}^{+}_{\omega}$

for programs and $\Psi \vdash \Omega \in F$ for cases. Regarding the judgments that are used in this Figure but not defined so far, $\Psi; \Gamma; \Delta \rhd M : A$ and $\Psi; \Gamma \rhd A : K$ are the standard LLF typing judgments, given in Appendix C. $\Psi; \chi' \vdash \chi \ : \Phi$ is a judgment that decides that χ is a valid context in world Φ, and it is defined in Appendix D. Also defined in Appendix D, $\Psi; \Gamma \rhd s : \text{sig}$ characterizes valid module signatures s. and $\Psi' \rhd \sigma : \Psi$ defines valid substitutions mapping objects valid in Ψ into objects valid in Ψ'.

The subordination relation on type families $A \prec: B$ is a relation suitable for narrowing, that decides if objects of A can occur as an index to B, and $A \hat{\prec} B$ is the related subordination relation deciding if objects of type A can occur in objects of type B, but never at the type level. For more information, consult [Vir99].

The first four rules in Figure 7 are introduction and elimination rules for the universal quantifiers. The fifth rule is the introduction rule for existentials. Existential elimination is a special case of the case rules defined in below the dotted line [Sch01], and need not be introduced individually. The typing rule for unit is standard. Ω, the argument to case, is a list of all of the cases (which must all have the same type). The type of a variable x can be inferred from the context, and recursion is standard. The rule for ν extends Ψ by a new declaration of a module signature, however, it must be guaranteed that α does not escape its scope, by requiring that the type of the body not contain the new declaration.

Unbounded recursion and a case construct without cases indicate, that without further side condition, $\mathcal{L}_\omega^+$ may contain partial and non-terminating functions. We attach therefore the following side conditions

1. for all $\Psi'' \triangleright \sigma' : \Psi$, there exists a $\Psi'' \triangleright \sigma'' : \Psi'$, such that $\sigma'' \circ \sigma = \sigma'$ (*)
2. and all occurrences of x in P terminate (**)

to the respective rules in Figure 7 that enforce totality. Syntactic criteria exist, but we cannot give them here in the interest of space.

5 Meta Theory of $\mathcal{L}_\omega^+$

That every function in $\mathcal{L}_\omega^+$ is total is a sufficient and necessary condition for the soundness of $\mathcal{L}_\omega^+$. The argument relies on a small-step operational semantics that is given in Appendix E. First, some notation. We define evaluation environments E to be those Ψ binding only block variables α. Next, we extend the set of programs with a closure $\{\sigma; P\}$, in which σ is a substitution that maps P from whatever environment it is well-typed under into the outer environment. The evaluation judgment $E \vdash P \to P'$ relates a program P to the outcome of a single evaluation step P'. For a sequence of zero or more evaluation steps, we write $E \vdash P \to^* P'$. The set of values is V.

$$V ::= \Lambda\, n :: (\chi \triangleright A).\ P \mid \Lambda\gamma \in (\chi \triangleright \Phi).\ P \mid \langle\!\langle \chi \triangleright M; V \rangle\!\rangle \mid \langle\!\langle\rangle\!\rangle$$

For functions, applications, existentials and fixed points, evaluation proceeds in the standard fashion. The evaluation of a closure $\{\sigma; P\}$ is essentially carrying out a single step of lazily applying the substitution σ to P. This is done because eager substitution is not sound in the presence of case. Evaluation of $(\text{case } \Omega, (\Psi \vdash \sigma' \mapsto P))$ in a closure proceeds by attempting to generate a substitution σ'' that, when composed with σ', is equivalent to the σ of the closure. If one is found, then evaluation of P continues in a closure under σ''. The evaluation of $\nu\ \alpha :: (\chi \triangleright s).\ P$ proceeds by evaluating P until it becomes a value. When it finally becomes a value, the ν binding is pushed into any non-values (as occur in a function) that may exist in the value. The following properties hold.

Theorem 2 (Type preservation). *If $E \vdash P \in F$ and $E \vdash P \to P'$ then $E \vdash P' \in F$.*

Proof. By induction on the structure of the evaluation relation. The cases for ν rely on the fact that the type of the body of the ν must not use the bound block variable. This allows the ν to be pushed inward while preserving the type. The substitution cases rely on the soundness of the substitution of σ into χ, A, M and x. □

Theorem 3 (Progress). *If $E \vdash P \in F$ then either P is a value or $E \vdash P \to P'$.*

Proof. By induction on the structure of the typing derivation. The progress proof uses the fact that E binds only block variables, and on the usual canonical forms lemma. It also relies on the coverage condition holding, which ensures that the program (case $\cdot$) is never evaluated. □

Theorem 4 (Termination). *If* $E \vdash P \in F$ *then* $E \vdash P \rightarrow^* V$.

Proof. By induction on the typing derivation, keeping track of the instantiations of the values bound by reductions of μ, using the termination condition. □

Theorem 5 (Soundness). *If* $\cdot \vdash P \in F$ *then* $\models F$.

Proof. By induction on F, using Theorems 2, 3 and 4. □

6 Conclusion

We have described the meta-logic $\mathcal{L}_\omega^+$ for the linear logical framework LLF. LLF is useful for the representation of formal systems that rely on a notion of deletable resource. Surprisingly many such systems can be represented in LLF, among them programming languages with effect, state transition system, such as the infamous blocks world often used in AI, and of course also resource oriented logics such as linear logic itself.

The meta-logic $\mathcal{L}_\omega^+$ is custom-made for LLF, which means, that incorporates knowledge about linear assumptions, how they are consumed, split in the multiplicative, and duplicated in the additive fragment. It enables the formalization of meta-theoretic properties, the mechanization of reasoning about LLF encodings, and leads to relatively short proof terms. The soundness of $\mathcal{L}_\omega^+$ follows from a realizability argument that shows that every function in $\mathcal{L}_\omega^+$ is total, i.e. it terminates and covers all cases.

In future work, we plan to implement a proof checker and an automated theorem prover for $\mathcal{L}_\omega^+$, and consider extensions to the ordered logical framework and the concurrent logical framework.

References

[App01] Andrew W. Appel. Foundational proof-carrying code. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 247–258, Boston, USA, June 2001.

[BCM00] David Basin, Manuel Clavel, and Jos Meseguer. Rewriting logic as a metalogical framework. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 55–80. Springer-Verlag LNCS 1974, 2000.

[CDL+99] Iliano Cervesato, Nancy Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A Meta-Notation for Protocol Analysis. In *12th Computer Security Foundations Workshop — CSFW-12*, pages 55–69, Mordano, Italy, 28–30 June 1999. IEEE Computer Society Press.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[CP96] Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science*, pages 264–275, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Har97] John Harrison. Floating point verification in HOL Light: The exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[McD97] Raymond McDowell. *Reasoning in a Logic with Definitions and Induction*. PhD thesis, University of Pennsylvania, 1997.

[MM97] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In Glynn Winskel, editor, *Proceedings of the Twelfth Annual Symposium on Logic in Computer Science*, pages 434–445, Warsaw, Poland, June 1997.

[NvO98] Tobias Nipkow and David von Oheimb. Java-light is type-safe — definitely. In L. Cardelli, editor, *Conference Record of the 25th Symposium on Principles of Programming Languages (POPL'98)*, pages 161–170, San Diego, California, January 1998. ACM Press.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer-Verlag LNCS 828, 1994.

[Pau97] Lawrence C. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, June 1997.

[Pfe94] Frank Pfenning. A structural proof of cut elimination and its representation in a logical framework. Technical Report CMU-CS-94-218, Department of Computer Science, Carnegie Mellon University, November 1994.

[Pfe99] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999. In preparation.

[PP03] Brigitte Pientka and Frank Pfenning. Optimizing higher-order pattern unification. In *CADE-19*, Miami Beach, Florida, July 2003. To appear.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.

[Sch01] Carsten Schürmann. Recursion for higher-order encodings. In Laurent Fribourg, editor, *Proceedings of the Conference on Computer Science Logic (CSL 2001)*, pages 585–599, Paris, France, August 2001. Springer Verlag LNCS 2142.

[VC02] Joseph C. Vanderwaart and Karl Crary. A simplified account of the metatheory of linear lf. *Electronic Notes in Theoretical Computer Science*, 70(2), 2002.

[Vir99] Roberto Virga. *Higher-Order Rewriting with Dependent Types*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 1999. Forthcoming.

A Admissibility of Cut: Informal Proof

Theorem 6 (Admissibility of cut). *If* $\mathcal{P} :: \Gamma_1 \Longrightarrow C, \Delta_1$ *and* $\mathcal{Q} :: \Gamma_2, C \Longrightarrow \Delta_2$ *then* $\mathcal{R} :: \Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2$.

Proof. by lexicographic structural induction on the subformula A and simultaneously on the $\mathcal{P}, \mathcal{Q}$.

Case: $\otimes$R with $\otimes$L.

$\mathcal{P} :: \Gamma_1', \Gamma_1'' \Longrightarrow A \otimes B, \Delta_1', \Delta_2''$	(by assumption)
$\mathcal{P}_1 :: \Gamma_1' \Longrightarrow A, \Delta_1'$	(by assumption)
$\mathcal{P}_2 :: \Gamma_1'' \Longrightarrow B, \Delta_2''$	(by assumption)
$\mathcal{Q} :: \Gamma_2, A \otimes B \Longrightarrow \Delta_2$	(by assumption)
$\mathcal{Q}_1 :: \Gamma_2, A, B \Longrightarrow \Delta_2$	(by assumption)
$\mathcal{R}_1 :: \Gamma_1', \Gamma_2, B \Longrightarrow \Delta_1', \Delta_2$	(by ind. hyp. on $\mathcal{P}_1, \mathcal{Q}_1$)
$\mathcal{R} :: \Gamma_1', \Gamma_1'', \Gamma_2 \Longrightarrow \Delta_1', \Delta_2'', \Delta_2$	(by ind. hyp. on $\mathcal{P}_2, \mathcal{R}_1$)

Case: ax with all other rules

$\mathcal{P} :: A \Longrightarrow A$	(by assumption)
$\mathcal{Q}, \mathcal{R} :: \Gamma_2, A \Longrightarrow \Delta_2$	(by assumption)

Case: All other rules with ax

$\mathcal{P}, \mathcal{R} :: \Gamma_1 \Longrightarrow A, \Delta_1$	(by assumption)
$\mathcal{Q} :: A \Longrightarrow A$	(by assumption)

Case: $\otimes$R with all other rules. Cut formula: left premiss.

$\mathcal{P} :: \Gamma_1', \Gamma_1'' \Longrightarrow A \otimes B, C, \Delta_1', \Delta_1''$	(by assumption)
$\mathcal{P}_1 :: \Gamma_1' \Longrightarrow A, C, \Delta_1'$	(by assumption)
$\mathcal{P}_2 :: \Gamma_1'' \Longrightarrow B, \Delta_1''$	(by assumption)
$\mathcal{Q} :: \Gamma_2, C \Longrightarrow \Delta_2$	(by assumption)
$\mathcal{R}_1 :: \Gamma_1', \Gamma_2 \Longrightarrow A, \Delta_1', \Delta_2$	(by ind. hyp. on $\mathcal{P}_1, \mathcal{Q}$)
$\mathcal{R} :: \Gamma_1', \Gamma_1'', \Gamma_2 \Longrightarrow A \otimes B, \Delta_1', \Delta_1'', \Delta_2$	(by $\otimes$R on $\mathcal{R}_1, \mathcal{P}_2$)

Case: $\otimes$R with all other rules. Cut formula: right premiss.

$\mathcal{P} :: \Gamma_1', \Gamma_1'' \Longrightarrow A \otimes B, C, \Delta_1', \Delta_1''$	(by assumption)
$\mathcal{P}_1 :: \Gamma_1' \Longrightarrow A, \Delta_1'$	(by assumption)
$\mathcal{P}_2 :: \Gamma_1'' \Longrightarrow B, C, \Delta_1''$	(by assumption)
$\mathcal{Q} :: \Gamma_2, C \Longrightarrow \Delta_2$	(by assumption)
$\mathcal{R}_2 :: \Gamma_1'', \Gamma_2 \Longrightarrow B, \Delta_1'', \Delta_2$	(by ind. hyp. on $\mathcal{P}_2, \mathcal{Q}$)
$\mathcal{R} :: \Gamma_1', \Gamma_1'', \Gamma_2 \Longrightarrow A \otimes B, \Delta_1', \Delta_1'', \Delta_2$	(by $\otimes$R on $\mathcal{P}_1, \mathcal{R}_2$)

Case: $\otimes$L with all other rules.

$\mathcal{P} :: \Gamma_1', A \otimes B \Longrightarrow C, \Delta_1'$	(by assumption)
$\mathcal{P}' :: \Gamma_1', A, B \Longrightarrow C, \Delta_1'$	(by assumption)
$\mathcal{Q} :: \Gamma_2, C \Longrightarrow \Delta_2$	(by assumption)
$\mathcal{R}' :: \Gamma_1', \Gamma_2, A, B \Longrightarrow \Delta_1', \Delta_2$	(by ind. hyp. on $\mathcal{P}', \mathcal{Q}$)
$\mathcal{R} :: \Gamma_1', \Gamma_2, A \otimes B \Longrightarrow \Delta_1', \Delta_2$	(by $\otimes$L on $\mathcal{R}'$)

B Admissibility of Cut: Formal Proof

$$
\begin{array}{l}
\textbf{dec}\ \mathrm{ca} : \forall\gamma_1 \in \Phi.\, \forall\gamma_2 \in \Phi.\, \forall(C : (\cdot \triangleright \mathrm{o})).\\
\qquad\qquad \forall(P : (\gamma_1 \triangleright \mathrm{pos}\ C \to \#)).\, \forall(Q : (\gamma_2 \triangleright \mathrm{neg}\ C \to \#)).\\
\qquad\qquad \exists(R : (\gamma_1, \gamma_2 \triangleright \#)).\, \top
\end{array}
$$

$$
\begin{array}{l}
\textbf{fun}\ \mathrm{ca}\ (\gamma_1', \gamma_1'')\ \gamma_2\ (A \otimes B)\ (\hat{\lambda}p{:}\mathrm{pos}\ (A \otimes B).\, \mathrm{tensorR}\hat{}\, P_1\hat{}\, P_2\hat{}\, p)\\
\qquad (\hat{\lambda}n{:}\mathrm{neg}\ (A \otimes B).\, \mathrm{tensorL}\hat{}\, (\hat{\lambda}n_1{:}\mathrm{neg}\ A.\, \hat{\lambda}n_2{:}\mathrm{neg}\ B.\, Q_1\hat{}\, n_1\hat{}\, n_2)\hat{}\, n) =\\
\quad \textbf{new}\\
\qquad \alpha :: (\gamma_1', \gamma_2 \triangleright \exists n : \mathrm{neg}\ B.\, \epsilon)\\
\quad \textbf{in}\\
\qquad \textbf{let}\\
\qquad\quad \textbf{val}\ \langle R_1, \langle\rangle\rangle = \mathrm{ca}\ \gamma_1'\ (\gamma_2, \alpha : \exists n : \mathrm{neg}\ B.\, \epsilon)\ A\ P_1\\
\qquad\qquad\qquad\qquad (\hat{\lambda}n{:}\mathrm{neg}\ A.\, Q_1\hat{}\, n_1\hat{}\, \pi_{\mathrm{t}}(\alpha))\\
\qquad\quad \textbf{val}\ \langle R, \langle\rangle\rangle = \mathrm{ca}\ \gamma_1''\ (\gamma_1', \gamma_2)\ B\ P_2\\
\qquad\qquad\qquad\qquad (\hat{\lambda}n{:}\mathrm{neg}\ B.\, R_1[n/\pi_{\mathrm{t}}(\alpha)])\\
\qquad \textbf{in}\\
\qquad\quad \langle R, \langle\rangle\rangle\\
\qquad \textbf{end}\\
\quad \textbf{end}\\
\quad |\, \mathrm{ca}\ (\alpha : \exists n : \mathrm{neg}\ A.\, \epsilon)\ \gamma_2\ C\ (\hat{\lambda}p{:}\mathrm{pos}\ C.\, \mathrm{ax}\hat{}\, \pi_{\mathrm{t}}(\alpha)\hat{}\, p)\ (\hat{\lambda}n{:}\mathrm{neg}\ C.\, Q\hat{}\, n)\\
\qquad = \langle Q\hat{}\, (\pi_{\mathrm{t}}(\alpha)), \langle\rangle\rangle\\
\quad |\, \mathrm{ca}\ \gamma_1\ (\alpha : \exists p : \mathrm{pos}\ A.\, \epsilon)\ C\ (\hat{\lambda}p{:}\mathrm{pos}\ C.\, P\hat{}\, p)\ (\hat{\lambda}n{:}\mathrm{neg}\ C.\, \mathrm{ax}\hat{}\, n\hat{}\, \pi_{\mathrm{t}}(\alpha))\\
\qquad = \langle P\hat{}\, (\pi_{\mathrm{t}}(\alpha)), \langle\rangle\rangle\\
\quad |\, \mathrm{ca}\ (\gamma_1', \gamma_1'', \alpha : \exists p : \mathrm{pos}\ (A \otimes B).\, \epsilon)\ \gamma_2\ C\\
\qquad (\hat{\lambda}p{:}\mathrm{pos}\ C.\, \mathrm{tensorR}\hat{}\, (\hat{\lambda}p'{:}\mathrm{pos}\ A.\, P_1\hat{}\, p\hat{}\, p')\ \hat{}\, P_2\hat{}\, (\pi_{\mathrm{t}}(\alpha)))\ Q =\\
\quad \textbf{new}\\
\qquad \alpha' :: (\gamma_1', \gamma_2 \triangleright \exists p : \mathrm{pos}\ A.\, \epsilon)\\
\quad \textbf{in}\\
\qquad \textbf{let}\\
\qquad\quad \textbf{val}\ \langle R_1, \langle\rangle\rangle = \mathrm{ca}\ (\gamma_1', \alpha' : \exists p : \mathrm{pos}\ A.\, \epsilon)\ \gamma_2\ C\\
\qquad\qquad\qquad\qquad (\hat{\lambda}p{:}\mathrm{pos}\ C.\, P_1\hat{}\, p\hat{}\, (\pi_{\mathrm{t}}(\alpha')))\ Q\\
\qquad \textbf{in}\\
\qquad\quad \langle \mathrm{tensorR}\hat{}\, (\hat{\lambda}p{:}\mathrm{pos}\ A.\, R_1[p/(\pi_{\mathrm{t}}(\alpha'))])\hat{}\, P_2\hat{}\, (\pi_{\mathrm{t}}(\alpha))), \langle\rangle\rangle\\
\qquad \textbf{end}\\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
\mid \mathrm{ca}\ (\gamma_1', \gamma_1'', \alpha : \exists p : \mathrm{pos}\ (A \otimes B).\, \epsilon)\ \gamma_2\ C \\
\quad (\hat{\lambda} p{:}\mathrm{pos}\ C.\, \mathrm{tensorR}\hat{}\, P_1\hat{}\,(\hat{\lambda} p'{:}\mathrm{pos}\ B.\, P_2\hat{}\, p\hat{}\, p')\hat{}\,(\pi_{\mathrm{t}}(\alpha)))\ Q = \\
\quad \textbf{new} \\
\quad\quad \alpha' :: (\gamma_1'', \gamma_2 \triangleright \exists p : \mathrm{pos}\ B.\, \epsilon) \\
\quad \textbf{in} \\
\quad\quad \textbf{let} \\
\quad\quad\quad \textbf{val}\ \langle R_2, \langle\rangle\rangle = \mathrm{ca}\ (\gamma_1'', \alpha' : \exists p : \mathrm{pos}\ B.\, \epsilon)\ \gamma_2\ C \\
\quad\quad\quad\quad\quad\quad\quad\quad (\hat{\lambda} p{:}\mathrm{pos}\ C.\, P_2\hat{}\, p\hat{}\,(\pi_{\mathrm{t}}(\alpha')))\ Q \\
\quad\quad \textbf{in} \\
\quad\quad\quad \langle \mathrm{tensorR}\hat{}\, P_1\hat{}\,(\hat{\lambda} p{:}\mathrm{pos}\ A.\, R_2[p/\pi_{\mathrm{t}}(\alpha')])\hat{}\,(\pi_{\mathrm{t}}(\alpha))), \langle\rangle\rangle \\
\quad\quad \textbf{end} \\
\quad \textbf{end} \\
\mid \mathrm{ca}\ (\gamma_1, \alpha : \exists n : \mathrm{neg}\ (A \otimes B).\, \epsilon)\ \gamma_2\ C \\
\quad (\hat{\lambda} p{:}\mathrm{pos}\ C.\, \mathrm{tensorL}\hat{}\,(\hat{\lambda} n_1{:}\mathrm{neg}\ A.\, \hat{\lambda} n_2{:}\mathrm{neg}\ B. \\
\quad\quad\quad P'\hat{}\, p\hat{}\, n_1\hat{}\, n_2)\hat{}\,(\pi_{\mathrm{t}}(\alpha)))\ Q = \\
\quad \textbf{new} \\
\quad\quad \alpha_1 :: (\gamma_1, \gamma_2 \triangleright \exists n : \mathrm{neg}\ A.\, \epsilon) \\
\quad\quad \alpha_2 :: (\gamma_1, \gamma_2, \alpha_1 : \exists n : \mathrm{neg}\ A.\, \epsilon \triangleright \exists n : \mathrm{neg}\ B.\, \epsilon) \\
\quad \textbf{in} \\
\quad\quad \textbf{let} \\
\quad\quad\quad \textbf{val}\ \langle R', \langle\rangle\rangle = \mathrm{ca}\ (\gamma_1, \alpha_1 : \exists n : \mathrm{neg}\ A.\, \epsilon, \alpha_2 : \exists n : \mathrm{neg}\ A.\, \epsilon)\ \gamma_2\ C \\
\quad\quad\quad\quad\quad\quad\quad\quad (\hat{\lambda} p{:}\mathrm{pos}\ C.\, P'\hat{}\, p\hat{}\,(\pi_{\mathrm{t}}(\alpha_1))\hat{}\,(\pi_{\mathrm{t}}(\alpha_2))\ Q) \\
\quad\quad \textbf{in} \\
\quad\quad\quad \langle \mathrm{tensorL}\hat{}\,(\hat{\lambda} n_1{:}\mathrm{neg}\ A.\, \hat{\lambda} n_2{:}\mathrm{neg}\ B. \\
\quad\quad\quad\quad R'[n_1/\pi_{\mathrm{t}}(\alpha_1), n_2/\pi_{\mathrm{t}}(\alpha_2)])\hat{}\,(\pi_{\mathrm{t}}(\alpha)); \langle\rangle\rangle \\
\quad\quad \textbf{end} \\
\quad \textbf{end}
\end{array}
$$

C Typing in LLF

C.1 Kind Formation

Judgment form: $\Psi; \Gamma \triangleright K : \mathrm{kind}$

$$\frac{}{\Psi; \Gamma \triangleright \mathrm{type} : \mathrm{kind}} \qquad \frac{\Psi; \Gamma \triangleright A : \mathrm{type} \quad \Psi; \Gamma, u : A \triangleright K : \mathrm{kind}}{\Psi; \Gamma \triangleright \Pi u : A.\ K : \mathrm{kind}}$$

C.2 Type Typing

Judgment form: $\Psi; \Gamma \triangleright A : K$

$$\frac{}{\Psi; \Gamma \triangleright a : \Sigma(a)} \qquad \frac{\Psi; \Gamma \triangleright A : \Pi u : A.\ K \quad \Psi; \Gamma; \cdot \triangleright M : A}{\Psi; \Gamma \triangleright A\ M : [\mathrm{id}_\Gamma, M/u]K}$$

$$\frac{\Psi; \Gamma \triangleright A : \mathrm{type} \quad \Psi; \Gamma, u : A \triangleright B : \mathrm{type}}{\Psi; \Gamma \triangleright \Pi u : A.\ B : \mathrm{type}} \qquad \frac{\Psi; \Gamma \triangleright A : \mathrm{type} \quad \Psi; \Gamma \triangleright B : \mathrm{type}}{\Psi; \Gamma \triangleright A \multimap B : \mathrm{type}}$$

$$\frac{\Psi; \Gamma \triangleright A : \mathrm{type} \quad \Psi; \Gamma \triangleright B : \mathrm{type}}{\Psi; \Gamma \triangleright A\ \&\ B : \mathrm{type}} \qquad \frac{}{\Psi; \Gamma \triangleright \top : \mathrm{type}}$$

C.3 Object Typing

Judgment form: $\Psi; \Gamma; \Delta \triangleright M : A$

$$\frac{\Psi(n) = (\chi \triangleright A) \quad \Psi; \Gamma'; \Delta' \triangleright \rho : \llbracket \chi \rrbracket_A^{\prec}; \llbracket \chi \rrbracket_A^{\hat{\prec}}}{\Psi; \Gamma'; \Delta' \triangleright n[\rho] : [\rho]A}$$

$$\frac{}{\Psi; \Gamma; \cdot \triangleright c : \Sigma(c)} \qquad \frac{}{\Psi; \Gamma; \cdot \triangleright b : \Gamma(b)} \qquad \frac{}{\Psi; \Gamma; b : A \triangleright b : A}$$

$$\frac{\Psi; \Gamma, u : A; \Delta \triangleright M : B}{\Psi; \Gamma; \Delta \triangleright \lambda u : A.\ M : \Pi u : A.\ B} \qquad \frac{\Psi; \Gamma; \Delta \triangleright M : \Pi u : A.\ B \quad \Psi; \Gamma; \cdot \triangleright N : A}{\Psi; \Gamma; \Delta \triangleright M\ N : [\mathrm{id}_\Gamma, N/u]B}$$

$$\frac{\Psi; \Gamma; \Delta, u : A \triangleright M : B}{\Psi; \Gamma; \Delta \triangleright \hat{\lambda} u : A.\ M : A \multimap B} \qquad \frac{\Psi; \Gamma; \Delta_1 \triangleright M : A \multimap B \quad \Psi; \Gamma; \Delta_2 \triangleright N : A}{\Psi; \Gamma; \Delta_1, \Delta_2 \triangleright M\ \hat{}\ N : B}$$

$$\frac{\Psi; \Gamma; \Delta \triangleright M : A \Psi; \Gamma; \Delta \triangleright N : B}{\Psi; \Gamma; \Delta \triangleright \langle M, N \rangle : A\ \&\ B} \qquad \frac{\Psi; \Gamma; \Delta \triangleright M : A\ \&\ B}{\Psi; \Gamma; \Delta \triangleright \pi_1 M : A}$$

$$\frac{}{\Psi; \Gamma; \Delta \triangleright \langle\rangle : \top} \qquad \frac{\Psi; \Gamma; \Delta \triangleright M : A\ \&\ B}{\Psi; \Gamma; \Delta \triangleright \pi_2 M : B}$$

D $\mathcal{L}_\omega^+$ Typing

D.1 Module Signature Typing.

Judgment form: $\Psi; \Gamma \triangleright s : k$

$$\frac{}{\Psi; \Gamma \triangleright \epsilon : \mathrm{sig}}$$

$$\frac{\Psi; \Gamma \triangleright A : \mathrm{type} \quad \Psi; \Gamma, u : A \triangleright s : \mathrm{sig}}{\Psi; \Gamma \triangleright \exists u : A.\ s : \mathrm{sig}} \qquad \frac{\Psi; \Gamma \triangleright A : \mathrm{type} \quad \Psi; \Gamma, u : A \triangleright s : \mathrm{sig}}{\Psi; \Gamma \triangleright \lambda u : A.s : \Pi u : A.\ k}$$

D.2 World Formation.

Judgment form: $\vdash \Phi$ ok

$$\frac{\cdot; \cdot \triangleright s : k}{\vdash s\ \mathrm{ok}} \qquad \frac{\vdash \Phi\ \mathrm{ok}}{\vdash \Phi^*\ \mathrm{ok}} \qquad \frac{\vdash \Phi_1\ \mathrm{ok} \quad \vdash \Phi_2\ \mathrm{ok}}{\vdash \Phi_1 + \Phi_2\ \mathrm{ok}}$$

D.3 Block Context Formation.

Judgment form: $\Psi; \chi' \vdash \chi \ : \Phi$

$$\frac{}{\Psi; \chi \vdash \cdot \ : \Phi^*} \qquad \frac{\Psi(\gamma) = (\chi \triangleright \Phi)}{\Psi; \chi \vdash \cdot, (\gamma \in \Phi) \ : \Phi} \qquad \frac{\Psi; [\![\chi]\!] \triangleright s : \mathrm{sig}}{\Psi; \chi \vdash \cdot, (\alpha : s) \ : s}$$

$$\frac{\Psi; \chi \vdash \chi' \ : [M/u]w \quad \Psi; \cdot \triangleright A : \mathrm{type} \quad \Psi; [\![\chi]\!]; \cdot \triangleright M : A}{\Psi; \chi \vdash \chi' \ : \lambda u : A.w}$$

$$\frac{\Psi; \chi_0 \vdash \chi_1 \ : \Phi \quad \Psi; \chi_0, \chi_1 \vdash \chi_2 \ : \Phi^*}{\Psi; \chi_0 \vdash \chi_1, \chi_2 \ : \Phi^*}$$

$$\frac{\Psi; \chi \vdash \chi' \ : \Phi_1}{\Psi; \chi \vdash \chi' \ : \Phi_1 + \Phi_2} \qquad \frac{\Psi; \chi \vdash \chi' \ : \Phi_2}{\Psi; \chi \vdash \chi' \ : \Phi_1 + \Phi_2}$$

D.4 Formula validity

Judgment form: $\Psi \vdash F \ \mathrm{ok}$

$$\frac{\Psi; [\![\chi]\!]^{\prec:}_A \triangleright A : \mathrm{type} \quad \Psi, n :: (\chi \triangleright A) \vdash F \ \mathrm{ok} \quad \Psi; \cdot \vdash \chi \ : \Phi}{\Psi \vdash \forall \ n :: (\chi \triangleright A).\ F \ \mathrm{ok}}$$

$$\frac{\vdash \Phi \ \mathrm{ok} \quad \Psi, \gamma \in (\chi \triangleright \Phi) \vdash F \ \mathrm{ok} \quad \Psi; \cdot \vdash \chi \ : \Phi'}{\Psi \vdash \forall \gamma \in (\chi \triangleright \Phi).\ F \ \mathrm{ok}}$$

$$\frac{\Psi; [\![\chi]\!]^{\prec:}_A \triangleright A : \mathrm{type} \quad \Psi, n :: (\chi \triangleright A) \vdash F \ \mathrm{ok} \quad \Psi; \cdot \vdash \chi \ : \Phi}{\Psi \vdash \exists \ n :: (\chi \triangleright A).\ F \ \mathrm{ok}}$$

$$\frac{}{\Psi \vdash \top \ \mathrm{ok}}$$

D.5 Context formation

Judgment form: $\vdash \Psi \ \mathrm{ok}$

$$\frac{}{\vdash \cdot \ \mathrm{ok}} \qquad \frac{\vdash \Psi \ \mathrm{ok} \quad \Psi; [\![\chi]\!]^{\prec:}_A \triangleright A : \mathrm{type} \quad \Psi; \cdot \vdash \chi \ : \Phi}{\vdash \Psi, n :: (\chi \triangleright A) \ \mathrm{ok}} \qquad \frac{\vdash \Psi \ \mathrm{ok} \quad \Psi; \cdot \vdash \chi \ : \Phi \quad \vdash \Phi \ \mathrm{ok}}{\vdash \gamma \in (\chi \triangleright \Phi) \ \mathrm{ok}}$$

$$\frac{\vdash \Psi \ \mathrm{ok} \quad \Psi \vdash F \ \mathrm{ok}}{\vdash \Psi, x \in F \ \mathrm{ok}} \qquad \frac{\vdash \Psi \ \mathrm{ok} \quad \Psi; [\![\chi]\!] \triangleright s : \mathrm{sig} \quad \Psi; \cdot \vdash \chi \ : \Phi}{\vdash \Psi, \alpha :: (\chi \triangleright s) \ \mathrm{ok}}$$

D.6 Substitution wellformedness

Judgment form: $\Psi' \triangleright \sigma : \Psi$

$$\frac{}{\Psi' \triangleright \cdot : \cdot} \qquad \frac{\Psi' \triangleright \sigma : \Psi \quad \Psi'; [\![[\sigma]\chi]\!]^{\prec:}_{[\sigma]A}; [\![[\sigma]\chi]\!]^{\hat{\sim}}_{[\sigma]A} \triangleright M : [\sigma]A}{\Psi' \triangleright (\sigma, M/n) : (\Psi, n :: (\chi \triangleright A))}$$

$$\frac{\Psi' \triangleright \sigma : \Psi \quad \Psi'; [\sigma]\chi \vdash \chi' \ : \Phi}{\Psi' \triangleright (\sigma, \chi'/\gamma) : (\Psi, \gamma \in (\chi \triangleright \Phi))} \qquad \frac{\Psi' \triangleright \sigma : \Psi \quad \Psi' \vdash P \in [\sigma]F}{\Psi' \triangleright (\sigma, P/x) : \Psi, x \in F}$$

$$\frac{\Psi' \triangleright \sigma : \Psi \quad \Psi'; [\sigma]\chi \triangleright m : [\sigma]s}{\Psi' \triangleright (\sigma, m/\alpha) : \Psi, \alpha :: (\chi \triangleright s)}$$

E Operational Semantics

E.1 Congruence Rules

$$\frac{E \vdash P \to P'}{E \vdash P\ M \to P'\ M} \qquad \frac{E \vdash P \to P'}{E \vdash P\ \chi \to P'\ \chi} \qquad \frac{E \vdash P \to P'}{E \vdash \langle\!\langle \chi \triangleright M; P \rangle\!\rangle \to \langle\!\langle \chi \triangleright M; P' \rangle\!\rangle}$$

$$\frac{E, \alpha :: (\chi \triangleright s) \vdash P \to P'}{E \vdash \nu\ \alpha :: (\chi \triangleright s).\ P \to \nu\ \alpha :: (\chi \triangleright s).\ P'}$$

E.2 Enclosure rules

$$\frac{}{E \vdash (\Lambda\ n :: (\chi \triangleright A).\ P)\ M \to \{\mathrm{id}_E, M/n; P\}}$$

$$\frac{}{E \vdash (\Lambda\gamma \in (\chi \triangleright \Phi).\ P)\ \chi \to \{\mathrm{id}_E, \chi/\gamma; P\}}$$

$$\frac{}{E \vdash \mathrm{case}\ \Omega \to \{\mathrm{id}_E; \mathrm{case}\ \Omega\}} \qquad \frac{}{E \vdash \mu\ x \in F.\ P \to \{\mathrm{id}_E; \mu\ x \in F.\ P\}}$$

E.3 Substitution Rules

$$\frac{}{E \vdash \{\sigma; \Lambda\ n :: (\chi \triangleright A).\ P\} \to \Lambda\ n :: ([\sigma]\chi \triangleright [\sigma]A).\ \{\sigma, n/n; P\}}$$

$$\frac{}{E \vdash \{\sigma; \Lambda\gamma \in (\chi \triangleright \Phi).\ P\} \to \Lambda\gamma \in ([\sigma]\chi \triangleright \Phi).\ \{\sigma, \gamma/\gamma; P\}}$$

$$\frac{}{E \vdash \{\sigma; P\ M\} \to \{\sigma; P\}\ ([\sigma]M)} \qquad \frac{}{E \vdash \{\sigma; P\ \chi\} \to \{\sigma; P\}\ ([\sigma]\chi)}$$

$$\frac{}{E \vdash \{\sigma; \langle\!\langle \chi \triangleright M; P \rangle\!\rangle\} \to \langle\!\langle [\sigma]\chi \triangleright [\sigma]M; \{\sigma; P\} \rangle\!\rangle}$$

$$\frac{}{E \vdash \{\sigma; x\} \to [\sigma]x}$$

$$\frac{\sigma'' \circ \sigma' = \sigma}{E \vdash \{\sigma; \text{case}\ (\Omega, (\Psi \vdash \sigma' \mapsto P))\} \to \{\sigma''; P\}}$$

$$\frac{}{E \vdash \{\sigma; \text{case}\ (\Omega, (\Psi \vdash \sigma' \mapsto P))\} \to \{\sigma; \text{case}\ \Omega\}}$$

$$\frac{}{E \vdash \{\sigma; \mu\ x \in F.\ P\} \to \{\sigma, (\mu\ x \in F.\ P)/x; P\}}$$

$$\frac{}{E \vdash \{\sigma; \nu\ \alpha :: (\chi \triangleright s).\ P\} \to \nu\ \alpha :: ([\sigma]\chi \triangleright [\sigma]s).\ \{\sigma, \alpha/\alpha; P\}}$$

$$\frac{}{E \vdash \{\sigma; \langle\!\langle\rangle\!\rangle\} \to \langle\!\langle\rangle\!\rangle} \qquad \frac{}{E \vdash \{\sigma; \{\sigma'; P\}\} \to \{\sigma \circ \sigma'; P\}}$$

E.4 Block Abstraction Rules

$$\frac{}{E \vdash \nu\ \alpha :: (\chi \triangleright s).\ \langle\!\langle\rangle\!\rangle \to \langle\!\langle\rangle\!\rangle}$$

$$\frac{}{E \vdash \nu\ \alpha :: (\chi \triangleright s).\ \langle\!\langle \chi' \triangleright M; P \rangle\!\rangle \to \langle\!\langle \chi' \triangleright M; \nu\ \alpha :: (\chi \triangleright s).\ P \rangle\!\rangle}$$

$$\frac{}{E \vdash \nu\ \alpha :: (\chi \triangleright s).\ \Lambda\ n :: (\chi' \triangleright A).\ P \to \Lambda\ n :: (\chi' \triangleright A).\ \nu\ \alpha :: (\chi \triangleright s).\ P}$$

$$\frac{}{E \vdash \nu\ \alpha :: (\chi \triangleright s).\ \Lambda\gamma \in (\chi' \triangleright \Phi).\ P \to \Lambda\gamma \in (\chi' \triangleright \Phi).\ \nu\ \alpha :: (\chi \triangleright s).\ P}$$