Structural Cut Elimination in Linear Logic

Frank Pfenning

December 1994

CMU-CS-94-222

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a new proof of cut elimination for linear logic which proceeds by three nested structural inductions, avoiding the explicit use of multi-sets and termination measures on sequent derivations. The computational content of this proof is a non-deterministic algorithm for cut elimination which is amenable to an elegant implementation in Elf. We show this implementation in detail.

This work was supported by NSF Grant CCR-9303383

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. government.

Keywords: Linear Logic, Cut Elimination, Logical Framework

Contents

1 Introduction

The property of cut elimination [Gen35] is a central property of logical systems in a sequent formulation. It usually yields consistency as an easy corollary and forms the basis for efficient proof search procedures [Wal90] and logic programming languages [MNPS91]. In linear logic, the algorithm for cut elimination may also be given a computational interpretation [Abr93]. Traditional sequent formulations of linear logic [Gir87] are not well-suited to a deep analysis of cut elimination because of the disturbing effects of weakening and contraction of modal formulas of the form $!A$ and $?A$. This led to the discovery of proof nets, but the question of cut elimination for a more syntactic sequent calculus remains important for applications of linear logic in computer science.

In the literature one can find formulations of linear logic in which applications of structural rules are more controlled than in CLL, a system with explicit weakening, contraction, and dereliction for modal formulas. This is achieved by dividing a sequent into linear and non-linear zones whose constituents are treated differently in the sequent rules. Examples of such calculi are Andreoli's Σ_2 [And92], Girard's LU [Gir93], and Hodas & Miller's $\mathcal{L}$ [HM94, Hod94]. We take a two-sided version of classical linear logic quite close to Σ_2 with three rules of Cut as in LU and endow the resulting calculus LV with proof terms. We then prove admissibility of the cut rules in LV by three nested structural inductions. Cut elimination follows by an additional structural induction on sequent derivations possibly containing cuts. The seeds for this proof may already be found in [Hod94], where cut elimination is proven for $\mathcal{L}$, a fragment of intuitionistic linear logic.

We show how linear sequent derivations may be represented faithfully in the logical framework LF [HHP93] and how this representation may be improved if the framework itself already possesses linear features. We also present an elegant implementation of the algorithm for cut elimination implicit in our constructive proof of the cut elimination theorem. This implementation is in Elf [Pfe91], a constraint logic programming language based on LF.

2 CLL: A Traditional Sequent Calculus for Linear Logic

In this section we present a traditional sequent calculus CLL for linear logic. There are so-called one-sided formulations (in which the involutive negation is defined except on atomic formulas) and two-sided formulation, where negation is sometimes defined via linear implication and falsehood. Here we chose a slightly different two-sided presentation, where negation is primitive, but where we restrict ourselves to one multiplicative and additive connective and constant. This is sufficient to define all other connectives; taking them as primitive would only add tedium, but nothing essentially new to the calculus. We use x and y for variables, a for parameters (that is, variables occuring free in derivations), t for first-order terms, and P for atomic formulas $p(t_1, \ldots, t_n)$.

$$\begin{array}{lrcl} \text{Formulas} & A & ::= & P \mid A_1 \otimes A_2 \mid 1 \mid A_1 \& A_2 \mid \top \mid A^\perp \mid \forall x.\ A \mid !A \mid ?A \\ \text{Sequences} & \Gamma & ::= & \cdot \mid \Gamma, A \end{array}$$

We use A, B, and C for formulas, Γ, Δ and later Ψ and Θ for sequences of formulas. We overload the comma and write $\Gamma = \Gamma', A$ or $\Gamma = A, \Gamma'$ if Γ is the result of adding one occurrence of A at any position in Γ'. Similarly, $\Gamma = \Gamma_1, \Gamma_2$ consists of all formula occurrences from Γ_1 and Γ_2. We also omit a leading $\cdot$ for the sake of brevity. Another way of expressing these conventions is to state that we treat sequences Γ of formulas as multi-sets. A sequent of CLL has the form $\Gamma \Longrightarrow \Delta$. We refer to formulas in Γ as *hypotheses* or *negative formulas*, to those in Δ as *conclusions* or *positive formulas*. We consider a sequent a judgment defined by the following inferences rules. We avoid explicit exchange rules by the conventions stated above.

Axioms.

$$\frac{}{A \Longrightarrow A}\, I$$

Multiplicative Connectives.

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

$$\frac{}{\cdot \Longrightarrow 1}\, 1R \qquad \frac{\Gamma \Longrightarrow \Delta}{\Gamma, 1 \Longrightarrow \Delta}\, 1L$$

Additive Connectives.

$$\frac{\Gamma \Longrightarrow A, \Delta \qquad \Gamma \Longrightarrow B, \Delta}{\Gamma \Longrightarrow A \& B, \Delta}\, \& R \qquad \begin{array}{c} \dfrac{\Gamma, A \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}\, \& L_1 \\[2ex] \dfrac{\Gamma, B \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}\, \& L_2 \end{array}$$

$$\frac{}{\Gamma \Longrightarrow \top, \Delta}\, \top R \qquad \text{No } \top L \text{ rule}$$

Involution.

$$\frac{\Gamma, A \Longrightarrow \Delta}{\Gamma \Longrightarrow A^{\perp}, \Delta}\, \neg R \qquad \frac{\Gamma \Longrightarrow A, \Delta}{\Gamma, A^{\perp} \Longrightarrow \Delta}\, \neg L$$

Quantification.

$$\frac{\Gamma \Longrightarrow [a/x]A, \Delta}{\Gamma \Longrightarrow \forall x.\, A, \Delta}\, \forall R^a \qquad \frac{\Gamma, [t/x]A \Longrightarrow \Delta}{\Gamma, \forall x.\, A \Longrightarrow \Delta}\, \forall L$$

Exponentials.

$$\frac{!\Gamma \Longrightarrow A, ?\Delta}{!\Gamma \Longrightarrow !A, ?\Delta}\, !R \qquad \frac{\Gamma, A \Longrightarrow \Delta}{\Gamma, !A \Longrightarrow \Delta}\, !L$$

$$\frac{\Gamma \Longrightarrow A, \Delta}{\Gamma \Longrightarrow ?A, \Delta}?R \qquad \frac{!\Gamma, A \Longrightarrow ?\Delta}{!\Gamma, ?A \Longrightarrow ?\Delta}?L$$

Structural Rules.

$$\frac{\Gamma \Longrightarrow \Delta}{\Gamma, !A \Longrightarrow \Delta}W! \qquad \frac{\Gamma \Longrightarrow \Delta}{\Gamma \Longrightarrow ?A, \Delta}W?$$

$$\frac{\Gamma, !A, !A \Longrightarrow \Delta}{\Gamma, !A \Longrightarrow \Delta}C! \qquad \frac{\Gamma \Longrightarrow ?A, ?A, \Delta}{\Gamma \Longrightarrow ?A, \Delta}C?$$

In the rule $\forall R$ the parameter a must be new, that is, it may not occur in Γ, $\forall x.\ A$, or Δ. The notation $!\Gamma$ and $?\Delta$ In the $!R$ and $?L$ rules expresses the side conditions that all formulas in these sequences must have the form $!B$ and $?C$, respectively. The rule of Cut below turns out to be admissible and thus does not need to be included as a primitive. We eventually establish this by a detour via the system LV introduced in the next section.

Cut.

$$\frac{\Gamma_1 \Longrightarrow A, \Delta_1 \qquad \Gamma_2, A \Longrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}\text{Cut}$$

Exploiting the symmetries of classical linear logic, we can define the remaining linear connectives that are ordinarily used.

$$\begin{array}{rcl} A \multimap B & \equiv & (A \otimes B^\perp)^\perp \\ A \parr B & \equiv & (A^\perp \otimes B^\perp)^\perp \\ \perp & \equiv & 1^\perp \\ A \oplus B & \equiv & (A^\perp \& B^\perp)^\perp \\ 0 & \equiv & \top^\perp \\ \exists x.\ A & \equiv & (\forall x.\ A^\perp)^\perp \end{array}$$

3 LV: Another Sequent Calculus for Linear Logic

In this section we present a formulation of a sequent calculus for linear logic, called LV very close to Andreoli's dyadic system Σ_2 [And92] and similar to Girard's LU [Gir93]. It may also be considered a complete classical analogue of Hodas & Miller's $\mathcal{L}$ [HM94, Hod94], a formulation of a fragment of intuitionistic linear logic. LV will be amenable to a structural proof of cut elimination following ideas from an analysis of intuitionistic and classical sequent calculi [Pfe94]. Andreoli's goal is to study a paradigm of concurrent computation as search for restricted linear derivations, so he does not endow Σ_2 with proof terms, nor does he prove cut elimination for it. However, he analyzes its relationship to CLL. Hodas gives an explicit proof of cut elimination for $\mathcal{L}$ in [Hod94], which contains many of the same basic ideas we employ here. However, his proof cannot be structural, since he does not introduce proof terms.

Constructive cut elimination in a sequent formulation is difficult due to the nature of the structural rules, especially contraction. Girard [Gir87] uses proof nets (instead of a sequent calculus) for his strong normalization theorem partly for that reason. Galmiche and Perrier [GP] give a syntactic analysis of permutabilities of rules in a sequent calculus and apply it to cut elimination; our own analysis does not go quite as far, but we have a simpler proof of cut elimination. We conjecture that their presentation could be streamlined using our linear sequent calculus.

The main challenge is to isolate the non-linear reasoning and the associated structural rules. Assume we attempt to prove that the conclusion of the cut rule has a cut-free derivation by nested inductions on the size of the cut formula and the sizes of the cut-free derivations of the premises. In the case where the cut formula is contracted we encounter some difficulties.

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma_1 \Longrightarrow !A, \Delta_1\end{array} \qquad \dfrac{\begin{array}{c}\mathcal{E}_1\\ \Gamma_2, !A, !A \Longrightarrow \Delta_2\end{array}}{\Gamma_2, !A \Longrightarrow \Delta_2}\,C!}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}\,\mathrm{Cut}$$

It follows from the induction hypothesis that one copy of $!A$ can be eliminated from $\mathcal{E}_1$ by a cut of $\mathcal{D}$ and $\mathcal{E}_1$ yielding a cut-free derivation of

$$\begin{array}{c}\mathcal{E}_1'\\ \Gamma_1, \Gamma_2, !A \Longrightarrow \Delta_1, \Delta_2,\end{array}$$

but how do we proceed from here? We cannot apply the induction hypothesis to $\mathcal{D}$ and $\mathcal{E}_1'$, since the size of the cut formula $!A$ is still the same, but $\mathcal{E}_1'$ may be much larger than $\mathcal{E}_1$. Moreover, due to the multiplicative nature of the cut, we would obtain a derivation of $\Gamma_1, \Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_1, \Delta_2$ and not of $\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2$ as needed. Since contraction is not valid in general, but only for modal formulas, this second problem cannot be easily repaired as in the case of classical logic.

The solution is to eliminate the structural rules (in the example above contraction) as much as possible. To this end we divide a sequent into linear and non-linear zones as in Andreoli's Σ_2 [And92] and Girard's LU [Gir93]. Contraction and weakening are no longer required, since the non-linear zones are maintained monotonically: In each inference rule arbitrary non-linear side formulas are permitted and copied to all premises, thus eliminating the need for contraction. Moreover, arbitrary non-linear side formulas are permitted for axioms, eliminating the need for explicit weakening. What remains is a form of the structural rule of dereliction combined with contraction: We must be able to copy formulas in the non-linear zones into the linear zones. This copying is controlled enough to allow a structural proof of admissibility of cut (and cut elimination as a corollary).

We continue to use the following complete fragment of classical linear logic; the remaining connectives can easily be defined. P stands for atomic formulas.

$$\text{Formulas} \quad A \quad ::= \quad P \mid A_1 \otimes A_2 \mid 1 \mid A_1 \& A_2 \mid \top \mid A^\perp \mid \forall x.\, A \mid\, !A \mid\, ?A$$

A sequent has the form

$$\Psi; \Gamma \longrightarrow \Delta; \Theta$$

which may be interpreted as $!\Psi, \Gamma \longrightarrow \Delta, ?\Theta$ in ordinary linear sequent calculus. Thus the outer zones in the sequents represent non-linear hypotheses and conclusions, the inner zones must be treated linearly. Note that formulas of the form $!A$ and $?A$ may occur freely in Γ and Δ. We omit the non-linear zones Ψ and Θ if they are empty and simply write $\Gamma \longrightarrow \Delta$ for $\cdot; \Gamma \longrightarrow \Delta; \cdot$. On our connectives the calculus is defined by the following rules.

Axioms.

$$\frac{}{\Psi; A \longrightarrow A; \Theta}\, I$$

Multiplicative Connectives.

$$\frac{\Psi; \Gamma_1 \longrightarrow A, \Delta_1; \Theta \qquad \Psi; \Gamma_2 \longrightarrow B, \Delta_2; \Theta}{\Psi; \Gamma_1, \Gamma_2 \longrightarrow A \otimes B, \Delta_1, \Delta_2; \Theta}\, \otimes R \qquad \frac{\Psi; \Gamma, A, B \longrightarrow \Delta; \Theta}{\Psi; \Gamma, A \otimes B \longrightarrow \Delta; \Theta}\, \otimes L$$

$$\frac{}{\Psi; \cdot \longrightarrow 1; \Theta}\, 1R \qquad \frac{\Psi; \Gamma \longrightarrow \Delta; \Theta}{\Psi; \Gamma, 1 \longrightarrow \Delta; \Theta}\, 1L$$

Additive Connectives.

$$\frac{\Psi; \Gamma \longrightarrow A, \Delta; \Theta \qquad \Psi; \Gamma \longrightarrow B, \Delta; \Theta}{\Psi; \Gamma \longrightarrow A \& B, \Delta; \Theta}\, \& R \qquad \begin{array}{c} \dfrac{\Psi; \Gamma, A \longrightarrow \Delta; \Theta}{\Psi; \Gamma, A \& B \longrightarrow \Delta; \Theta}\, \& L_1 \\[2ex] \dfrac{\Psi; \Gamma, B \longrightarrow \Delta; \Theta}{\Psi; \Gamma, A \& B \longrightarrow \Delta; \Theta}\, \& L_2 \end{array}$$

$$\frac{}{\Psi; \Gamma \longrightarrow \top, \Delta; \Theta}\, \top R \qquad \text{No } \top L \text{ rule}$$

Involution.

$$\frac{\Psi; \Gamma, A \longrightarrow \Delta; \Theta}{\Psi; \Gamma \longrightarrow A^{\perp}, \Delta; \Theta}\, \neg R \qquad \frac{\Psi; \Gamma \longrightarrow A, \Delta; \Theta}{\Psi; \Gamma, A^{\perp} \longrightarrow \Delta; \Theta}\, \neg L$$

Quantification.

$$\frac{\Psi; \Gamma \longrightarrow [a/x]A, \Delta; \Theta}{\Psi; \Gamma \longrightarrow \forall x.\ A, \Delta; \Theta}\, \forall R^{a} \qquad \frac{\Psi; \Gamma, [t/x]A \longrightarrow \Delta; \Theta}{\Psi; \Gamma, \forall x.\ A \longrightarrow \Delta; \Theta}\, \forall L$$

Exponentials.

$$\frac{\Psi; \cdot \longrightarrow A; \Theta}{\Psi; \cdot \longrightarrow !A; \Theta}\, !R \qquad \frac{(\Psi, A); \Gamma \longrightarrow \Delta; \Theta}{\Psi; (\Gamma, !A) \longrightarrow \Delta; \Theta}\, !L$$

$$\frac{\Psi; \Gamma \longrightarrow \Delta; (A, \Theta)}{\Psi; \Gamma \longrightarrow (?A, \Delta); \Theta}\, ?R \qquad \frac{\Psi; A \longrightarrow \cdot; \Theta}{\Psi; ?A \longrightarrow \cdot; \Theta}\, ?L$$

Structural Rules.

$$\frac{(\Psi, A); (\Gamma, A) \longrightarrow \Delta; \Theta}{(\Psi, A); \Gamma \longrightarrow \Delta; \Theta}\,!D \qquad \frac{\Psi; \Gamma \longrightarrow (A, \Delta); (A, \Theta)}{\Psi; \Gamma \longrightarrow \Delta; (A, \Theta)}\,?D$$

In the rule $\forall R$ the parameter a must be new, that is, it may not occur in Ψ, Γ, $\forall x.\ A$, Δ, or Θ. There are three rules of cut, which we show to be admissible rather than taking them as primitive.

Cut.

$$\frac{\Psi; \Gamma_1 \longrightarrow A, \Delta_1; \Theta \qquad \Psi; \Gamma_2, A \longrightarrow \Delta_2; \Theta}{\Psi; \Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2; \Theta}\,\mathrm{Cut}$$

$$\frac{\Psi; \cdot \longrightarrow A; \Theta \qquad (\Psi, A); \Gamma \longrightarrow \Delta; \Theta}{\Psi; \Gamma \longrightarrow \Delta; \Theta}\,\mathrm{Cut!} \qquad \frac{\Psi; \Gamma \longrightarrow \Delta; (A, \Theta) \qquad \Psi; A \longrightarrow \cdot; \Theta}{\Psi; \Gamma \longrightarrow \Delta; \Theta}\,\mathrm{Cut?}$$

This system satisfies weakening and contraction in the non-linear zones: We can adjoin a formula to the non-linear zones of each sequent in a derivation to achieve weakening, or substitute the use of one formula for another to achieve contraction. In either case the structure of the derivation does not change.

Lemma 1 (Elementary Properties of LV)

1. *(*WL*) If* $\Psi; \Gamma \longrightarrow \Delta; \Theta$ *then* $(\Psi, A); \Gamma \longrightarrow \Delta; \Theta$*.*
2. *(*WR*) If* $\Psi; \Gamma \longrightarrow \Delta; \Theta$ *then* $\Psi; \Gamma \longrightarrow \Delta; (A, \Theta)$*.*
3. *(*CL*) If* $(\Psi, A, A); \Gamma \longrightarrow \Delta; \Theta$ *then* $(\Psi, A); \Gamma \longrightarrow \Delta; \Theta$*.*
4. *(*CR*) If* $\Psi; \Gamma \longrightarrow \Delta; (A, A, \Theta)$ *then* $\Psi; \Gamma \longrightarrow \Delta; (A, \Theta)$*.*

Proof: By simple inductions over the structure of the derivations of the assumption. Note that the constructed derivation differs from the one in the assumption only in that a non-linear hypothesis or conclusion has been added or erased. □

We can now show the equivalence of LV and CLL.

Theorem 2 (Equivalence of LV and CLL)

1. *If* $\Psi; \Gamma \longrightarrow \Delta; \Theta$ *then* $!\Psi, \Gamma \Longrightarrow \Delta, ?\Theta$*.*
2. *If* $!\Psi, \Gamma \Longrightarrow \Delta, ?\Theta$ *where formulas in* Γ *and* Δ *are not of the form* $!A$ *and* $?A$*, respectively, then* $\Psi; \Gamma \longrightarrow \Delta; \Theta$*.*

Proof: By structural inductions over the derivation in the assumption, using non-linear weakening and contraction from Lemma 1.

Soundness of LV. The proof proceeds by induction over the structure of $\mathcal{D} :: (\Psi;\Gamma \longrightarrow \Delta;\Theta)$. This is the simpler of the two directions. We only show a few cases; all others are similar.
Case:

$$\mathcal{D} = \frac{}{\Psi; A \longrightarrow A; \Theta} I.$$

Then

$I :: (A \Longrightarrow A)$ — Axiom
$\mathcal{E} :: (!\Psi, A \Longrightarrow A, ?\Theta)$ — By weakenings ($W!$ and $W?$) from I.

Case:

$$\mathcal{D} = \frac{\begin{matrix}\mathcal{D}_1 \\ \Psi;\Gamma_1 \longrightarrow A, \Delta_1; \Theta\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Psi;\Gamma_1 \longrightarrow B, \Delta_2; \Theta\end{matrix}}{\Psi;\Gamma_1,\Gamma_2 \longrightarrow A \otimes B, \Delta_1, \Delta_2; \Theta} \otimes R.$$

Then

$\mathcal{E}_1 :: (!\Psi, \Gamma_1 \Longrightarrow A, \Delta_1, ?\Theta)$ — By ind. hyp. on $\mathcal{D}_1$
$\mathcal{E}_2 :: (!\Psi, \Gamma_2 \Longrightarrow B, \Delta_2, ?\Theta)$ — By ind. hyp. on $\mathcal{D}_2$
$\mathcal{E}' :: (!\Psi, !\Psi, \Gamma_1, \Gamma_2 \Longrightarrow A \otimes B, \Delta_1, \Delta_2, ?\Theta, ?\Theta)$ — By $\otimes R$ from $\mathcal{E}_1$ and $\mathcal{E}_2$
$\mathcal{E} :: (!\Psi, \Gamma_1, \Gamma_2 \Longrightarrow A \otimes B, \Delta_1, \Delta_2, ?\Theta)$ — By contractions ($C!$ and $C?$) from $\mathcal{E}'$

Case:

$$\mathcal{D} = \frac{\begin{matrix}\mathcal{D}_1 \\ (\Psi, A);\Gamma, A \longrightarrow \Delta; \Theta\end{matrix}}{(\Psi, A);\Gamma \longrightarrow \Delta; \Theta} !D.$$

Then

$\mathcal{E}_1 :: (!\Psi, !A, \Gamma, A \Longrightarrow \Delta, ?\Theta)$ — By ind. hyp. on $\mathcal{D}_1$
$\mathcal{E}_1' :: (!\Psi, !A, \Gamma, !A \Longrightarrow \Delta, ?\Theta)$ — By $!L$ from $\mathcal{E}_1$
$\mathcal{E} :: (!\Psi, !A, \Gamma \Longrightarrow \Delta, ?\Theta)$ — By contraction ($C!$) from $\mathcal{E}_1'$

Completeness of LV. The proof of this direction proceeds by induction on the structure of $\mathcal{E} :: (!\Psi, \Gamma \Longrightarrow \Delta, ?\Theta)$. Recall that Γ may not contain a formula of the form $!A$, and Δ may not contain a formula of the form $?A$. We have to construct a $\mathcal{D} :: (\Psi;\Gamma \longrightarrow \Delta;\Theta)$. We only show a few cases; the others are similar.
Case:

$$\mathcal{E} = \frac{}{A \Longrightarrow A} I$$

There are three subcases, depending on whether the principal connective of A is !, ?, or neither. We show the case where $A = !A'$. We construct

$$\mathcal{D} = \cfrac{\cfrac{\cfrac{}{A'; A' \longrightarrow A'; \cdot} I}{A'; \cdot \longrightarrow A'; \cdot} D!}{A'; \cdot \longrightarrow !A'; \cdot} !R$$

to satisfy the requirement of the theorem.

Case:

$$\mathcal{E} = \dfrac{\begin{array}{cc} \mathcal{E}_1 & \mathcal{E}_2 \\ !\Psi_1, \Gamma_1 \Longrightarrow A, \Delta_1, ?\Theta_1 & !\Psi_2, \Gamma_2 \Longrightarrow B, \Delta_2, ?\Theta_2 \end{array}}{!\Psi_1, !\Psi_2, \Gamma_1, \Gamma_2 \Longrightarrow A \otimes B, \Delta_1, \Delta_2, ?\Theta_1, ?\Theta_2} \otimes R.$$

Then there are four similar subcases, depending on whether A and B are of the form $?A'$ and $?B'$, respectively. Assume that $A = ?A'$ and $B = ?B'$; the remaining subcases are similar. then

$\mathcal{D}_1 :: (\Psi_1; \Gamma_1 \longrightarrow \Delta_1; (A', \Theta_1))$	By ind. hyp. on $\mathcal{E}_1$
$\mathcal{D}_1' :: (\Psi_1; \Gamma_1 \longrightarrow A, \Delta_1; \Theta_1)$	By $?R$ from $\mathcal{D}_1$
$\mathcal{D}_1'' :: (\Psi_1, \Psi_2; \Gamma_1 \longrightarrow A, \Delta_1; \Theta_1, \Theta_2)$	By weakenings WL (Lemma 1) from $\mathcal{D}_1'$
$\mathcal{D}_2'' :: (\Psi_1, \Psi_2; \Gamma_2 \longrightarrow B, \Delta_2; \Theta_1, \Theta_2)$	Similarly from $\mathcal{E}_2$
$\mathcal{D} :: (\Psi_1, \Psi_2; \Gamma_1, \Gamma_2 \longrightarrow A \otimes B, \Delta_1, \Delta_2; \Theta_1, \Theta_2)$	By $\otimes R$ from $\mathcal{D}_1''$ and $\mathcal{D}_2''$.

□

We could prove admissibility of the cut rules in LV simulteneously by three nested inductions on the complexity of the cut formula, the length of the left derivation of the left premise and the length of the derivation of the right premise. The rule Cut would be considered smaller that Cut! and Cut? when the cut formula itself does not change. However, for the purpose of implementation we would like the proof to be structural. To this end we introduce proof terms to label derivations in the next section.

4 Linear Proof Terms

In this section we introduce proof terms so that the required weakening for formulas in the non-linear zones does not destroy the structural induction and to resolve ambiguities in the informal presentation (the formula labels track occurrences). It is helpful to think of proof terms as part of a *linear λ-calculus*. For our purposes, proof terms for the Lolli fragment of linear logic [HM94] are sufficient, which is important since it satisfies a stronger normal form theorem than the full calculus. In this fragment, we have linear ($\multimap$) and intuitionistic ($\rightarrow$) implication, additive conjunction ($\&$), top ($\top$) and corresponding proof constructors. Note that we define a calculus of *natural deduction* rather than a sequent calculus.

$$\begin{array}{llcll}
\text{Linear Types} & A & ::= & P \mid A_1 \multimap A_2 \mid A_1 \& A_2 \mid \top \mid A_1 \rightarrow A_2 & \\
\text{Linear Terms} & M & ::= & c \mid x & \\
 & & & \mid \lambda x{:}A.\ M \mid M_1\, M_2 & \text{for } A_1 \multimap A_2 \\
 & & & \mid \langle M_1, M_2 \rangle \mid \pi_1 M \mid \pi_2 M & \text{for } A_1 \& A_2 \\
 & & & \mid \langle\, \rangle & \text{for } \top \\
 & & & \mid \overset{\omega}{\lambda} x{:}A.\ M \mid M_1 \cdot M_2 & \text{for } A_1 \rightarrow A_2
\end{array}$$

We do not formally define this calculus here since it would lead us too far afield. Endowing a sequent derivation with a linear proof term, however, is suggestive for an encoding of LV in a linear logical framework that is more concise than the one we present here. We return to this issue in Section 8.

The proof terms must refer to the hypotheses and conclusions in a sequent in a unique way. We therefore label each formula in each zone. The proof term annotates the whole sequent, and, for lack of a better place, we write it above the sequent arrow.

$$(n_1^\omega{:}A_1, \ldots, n_j^\omega{:}A_j); (n_1{:}B_1, \ldots, n_k{:}B_k) \xrightarrow{d} (p_1{:}C_1, \ldots, p_l{:}C_l); (p_1^\omega{:}D_1, \ldots, p_m^\omega{:}D_m)$$

We systematically introduce exactly one proof term constructor for each inference rule and give it a mnemonic name. This avoids confusion with different proof term assignments from the literature (e.g. [Abr93]). The purpose of the proof terms here is not immediately computational—they record enough information about the structure of the proof that it may be reconstructed up to insertion or removal of some unused hypotheses or conclusions.

Axioms.

$$\frac{}{\Psi; n{:}A \xrightarrow{\mathrm{axiom}\,n\,p} p{:}A; \Theta}\, I$$

Multiplicative Connectives.

$$\frac{\Psi; \Gamma_1 \xrightarrow{d_1} p_1{:}A, \Delta_1; \Theta \qquad \Psi; \Gamma_2 \xrightarrow{d_2} p_2{:}B, \Delta_2; \Theta}{\Psi; \Gamma_1, \Gamma_2 \xrightarrow{\mathrm{timesr}\,(\lambda p_1{:}A.\ d_1)\,(\lambda p_2{:}B.\ d_2)\,p} p{:}A \otimes B, \Delta_1, \Delta_2; \Theta}\, \otimes R$$

$$\frac{\Psi; \Gamma, n_1{:}A, n_2{:}B \xrightarrow{d} \Delta; \Theta}{\Psi; \Gamma, n{:}A \otimes B \xrightarrow{\mathrm{timesl}\,(\lambda n_1{:}A.\ \lambda n_2{:}B.\ d)\,n} \Delta; \Theta}\, \otimes L$$

$$\frac{}{\Psi; \cdot \xrightarrow{\mathrm{oner}\,p} p{:}1; \Theta}\, 1R \qquad \frac{\Psi; \Gamma \xrightarrow{d} \Delta; \Theta}{\Psi; \Gamma, n{:}1 \xrightarrow{\mathrm{onel}\,d\,n} \Delta; \Theta}\, 1L$$

Additive Connectives.

$$\frac{\Psi; \Gamma \xrightarrow{d_1} p_1{:}A, \Delta; \Theta \qquad \Psi; \Gamma \xrightarrow{d_2} p_2{:}B, \Delta; \Theta}{\Psi; \Gamma \xrightarrow{\mathrm{andr}\,\langle(\lambda p_1{:}A.\ d_1), (\lambda p_2{:}B.\ d_2)\rangle\,p} p{:}A \& B, \Delta; \Theta}\, \& R$$

$$\frac{\Psi; \Gamma, n_1{:}A \xrightarrow{d_1} \Delta; \Theta}{\Psi; \Gamma, n{:}A \& B \xrightarrow{\mathrm{andl}_1\,(\lambda n_1{:}A.\ d_1)\,n} \Delta; \Theta}\, \& L_1$$

$$\frac{\Psi; \Gamma, n_2{:}B \xrightarrow{d_2} \Delta; \Theta}{\Psi; \Gamma, n{:}A \& B \xrightarrow{\mathrm{andl}_2\,(\lambda n_2{:}B.\ d_2)\,n} \Delta; \Theta}\, \& L_2$$

$$\frac{}{\Psi; \Gamma \xrightarrow{\mathrm{topr}\,\langle\rangle\,p} p{:}\top, \Delta; \Theta}\, \top R \qquad \text{No } \top L \text{ rule}$$

Involution.

$$\frac{\Psi; \Gamma, n{:}A \xrightarrow{d} \Delta; \Theta}{\Psi; \Gamma \xrightarrow{\mathrm{perpr}\,(\lambda n{:}A.\ d)\,p} p{:}A^\perp, \Delta; \Theta}\, \neg R \qquad \frac{\Psi; \Gamma \xrightarrow{d} p{:}A, \Delta; \Theta}{\Psi; \Gamma, n{:}A^\perp \xrightarrow{\mathrm{perpl}\,(\lambda p{:}A.\ d)\,n} \Delta; \Theta}\, \neg L$$

Quantification.

$$\frac{\Psi; \Gamma \xrightarrow{d} p_1{:}[a/x]A, \Delta; \Theta}{\Psi; \Gamma \xrightarrow{\text{forallr}\,(\lambda a{:}i.\ \lambda p_1{:}[a/x]A.\ d)\, p} p{:}\forall x.\ A, \Delta; \Theta}\,\forall R^a \qquad \frac{\Psi; \Gamma, n_1{:}[t/x]A \xrightarrow{d} \Delta; \Theta}{\Psi; \Gamma, n{:}\forall x.\ A \xrightarrow{\text{foralll}\, t\, (\lambda n_1{:}[t/x]A.\ d)\, n} \Delta; \Theta}\,\forall L$$

Exponentials.

$$\frac{\Psi; \cdot \xrightarrow{d} p_1{:}A; \Theta}{\Psi; \cdot \xrightarrow{!\text{r}\cdot(\lambda p_1{:}A.\ d)\, p} p{:}!A; \Theta}\,!R \qquad \frac{(\Psi, n^\omega{:}A); \Gamma \xrightarrow{d} \Delta; \Theta}{\Psi; (\Gamma, n{:}!A) \xrightarrow{!\text{l}\,(\lambda^\omega n^\omega{:}A.\ d)\, n} \Delta; \Theta}\,!L$$

$$\frac{\Psi; \Gamma \xrightarrow{d} \Delta; (p^\omega{:}A, \Theta)}{\Psi; \Gamma \xrightarrow{?\text{r}\,(\lambda^\omega p^\omega{:}A.\ d)\, p} (p{:}?A, \Delta); \Theta}\,?R \qquad \frac{\Psi; n_1{:}A \xrightarrow{d} \cdot; \Theta}{\Psi; n{:}?A \xrightarrow{?\text{l}\cdot(\lambda n_1{:}A.\ d)\, n} \cdot; \Theta}\,?L$$

Structural Rules.

$$\frac{(\Psi, n^\omega{:}A); (\Gamma, n{:}A) \xrightarrow{d} \Delta; \Theta}{(\Psi, n^\omega{:}A); \Gamma \xrightarrow{!\text{d}\,(\lambda n{:}A.\ d)\cdot n^\omega} \Delta; \Theta}\,!D \qquad \frac{\Psi; \Gamma \xrightarrow{d} (p{:}A, \Delta); (p^\omega{:}A, \Theta)}{\Psi; \Gamma \xrightarrow{?\text{d}\,(\lambda p{:}A.\ d)\cdot p^\omega} \Delta; (A, \Theta)}\,?D$$

A sequent derivation does not uniquely determine its proof term, nor does a proof term uniquely determine a derivation. The first phenomenon arises since sequent derivations without proof terms do not track occurrences. For example,

$$\dfrac{\dfrac{}{(A, A); A \longrightarrow A; \cdot}\,I}{(A, A); \cdot \longrightarrow A; \cdot}\,D!$$

when annotated as

$$(n_1^\omega{:}A, n_2^\omega{:}A); \cdot \xrightarrow{d} p{:}A; \cdot$$

would be well formed with either

$$\begin{aligned} d &= \ !\text{d}\,(\lambda n{:}A.\ \mathsf{axiom}\, n\, p) \cdot n_1^\omega, \quad \text{or} \\ d &= \ !\text{d}\,(\lambda n{:}A.\ \mathsf{axiom}\, n\, p) \cdot n_2^\omega, \end{aligned}$$

depending on whether the first or the second non-linear occurrence of A is copied to the linear zone. For the meta-theory of sequent calculi it is often assumed that we can "track occurrences"—proof terms provide a principled way to achieve this, since different occurrences of formulas have distinct labels.

In the other direction, a linear proof term does not uniquely determine its derivation. The first source of ambiguity already arises in the classical calculus: Adjoining modal hypotheses or modal conclusions to a derivation does not change the structure of its proof term. This is desirable, since it permits a proof of cut elimination by structural induction on the proof terms.

The second ambiguity is due to the additive nature of $\top$. Consider the two derivations

$$\dfrac{\dfrac{}{A \longrightarrow \top}\top R \qquad \dfrac{}{\cdot \longrightarrow \top}\top R}{A \longrightarrow \top \otimes \top}\otimes R \qquad \dfrac{\dfrac{}{\cdot \longrightarrow \top}\top R \qquad \dfrac{}{A \longrightarrow \top}\top R}{A \longrightarrow \top \otimes \top}\otimes R$$

The proof term d for the annotated sequent

$$n{:}A \xrightarrow{d} p{:}\top \otimes \top$$

in both cases is

$$\mathrm{timesr}\,(\lambda p_1{:}\top.\ \mathrm{topr}\,\langle\,\rangle\,p_1)\,(\lambda p_2{:}\top.\ \mathrm{topr}\,\langle\,\rangle\,p_2)\,p$$

Other hypotheses besides A are also permitted and could be split in different ways at the $\otimes R$ inference without affecting the proof term. Since proof terms forget information, it seems inevitable that our LF representation based on the proof terms above will not be adequate. The solution is to introduce a higher-level judgment that the representing terms are indeed well formed linear terms. The derivations that establish linearity of proof terms are then isomorphic to sequent derivations.

Despite these difficulties, we have the following properties of proof terms. Here we write $\mathcal{D}, v{:}A$ for the result of adjoining $v{:}A$ to the appropriate non-linear zone in every sequent in $\mathcal{D}$. Note that the proof term does not change when we adjoin a formula.

Lemma 3 (Basic Properties of Proof Terms)

1. *(*WL*) If* $\mathcal{D} :: (\Psi; \Gamma \xrightarrow{d} \Delta; \Theta)$ *then* $(\mathcal{D}, n^{\omega}{:}A) :: ((\Psi, n^{\omega}{:}A); \Gamma \xrightarrow{d} \Delta; \Theta)$.
2. *(*WR*) If* $\mathcal{D} :: (\Psi; \Gamma \xrightarrow{d} \Delta; \Theta)$ *then* $(\mathcal{D}, p^{\omega}{:}A) :: (\Psi; \Gamma \xrightarrow{d} \Delta; (p^{\omega}{:}A, \Theta))$.

Proof: In each case by a straightforward induction on the structure of $\mathcal{D}$. □

Of course, weakening in the linear zones of a sequent is not valid in general.

5 Representation in LF

In this section we develop the representation of linear sequent derivations in LF as implemented in Elf. We proceed in three stages. In the first stage we represent the formulas of linear logic using the standard technique of higher-order abstract syntax (see [HHP93]). In the second stage we encode proof terms for the sequent calculus without regard to linearity constraints as in [Pfe94]. In the third stage we encode the linearity requirements as a higher-level judgment on proof terms. This third stage yields an adequate representation.

5.1 Formulas

The representation of formulas using higher-order abstract syntax is straightforward. The main idea is to represent object-level variables and parameters by meta-level variables. Also, we do not commit to any particular domain of individuals, so only variables are built-in. One can encode constants or function symbols by declaring new LF constants of appropriate type `i -> ... -> i` and predicate symbols by declaring new LF constants of type `i -> ... -> o`. We only show the Elf declarations below. An explicit definition of the representation of A, $\ulcorner A \urcorner$, may be easily constructed from this signature. For example, $\ulcorner A^{\perp} \urcorner = \mathtt{perp}\ulcorner A \urcorner$ Note that we use readable infix notation for $\otimes$ and $\&$ which are considered right associative.

```
i : type.  % individuals
o : type.  % formulas

% Multiplicative connectives
times  : o -> o -> o.  %infix right 11 times
one    : o.

% Additive connectives
and    : o -> o -> o.  %infix right 11 and
top    : o.

% Involution
perp   : o -> o.

% Quantifier
forall : (i -> o) -> o.

% Exponentials
!      : o -> o.
?      : o -> o.
```

For example, $\ulcorner \forall x.\, (A\, x \& (B\, x)^{\perp}) \urcorner$ would be

```
(forall [x:i] A x and perp (B x)) : o.
```

in a context where `A:i -> o` and `B:i -> o`.

5.2 Proof Terms

The proof terms of the annotated sequent calculus are transcribed into LF, interpreting both λ and λ^{ω} as meta-level abstraction, and both juxtaposition and $\cdot$ as meta-level application. We use four distinct type families, `neg!`, `neg`, `pos`, and `pos?` to distinguish exponential and linear hypotheses and conclusions.

```
# : type.         % Token (for derivations)
neg!: o -> type.  % Modal Hypotheses (far left)
neg : o -> type.  % Hypotheses (left)
pos : o -> type.  % Conclusions (right)
pos?: o -> type.  % Modal Conclusions (far right).

axiom : (neg A -> pos A -> #).

timesr : (pos A -> #)
          -> (pos B -> #)
          -> (pos (A times B) -> #).

timesl : (neg A -> neg B -> #)
          -> (neg (A times B) -> #).

oner : (pos one -> #).

onel : # -> (neg one -> #).

andl1 : (neg A -> #)
         -> (neg (A and B) -> #).

andr : (pos A -> #) -> (pos B -> #)
        -> (pos (A and B) -> #).
```

```
andl2 : (neg B -> #)
         -> (neg (A and B) -> #).

topr : (pos (top) -> #).

% no topl

perpr : (neg A -> #)
         -> (pos (perp A) -> #).

perpl : (pos A -> #)
         -> (neg (perp A) -> #).

forallr : ({a:i} pos (A a) -> #)
           -> (pos (forall A) -> #).

foralll : {T:i} (neg (A T) -> #)
           -> (neg (forall A) -> #).

!r : (pos A -> #)
      -> (pos (! A) -> #).

!l : (neg! A -> #)
      -> (neg (! A) -> #).

?r : (pos? A -> #)
      -> (pos (? A) -> #).

?l : (neg A -> #)
      -> (neg (? A) -> #).

!d : (neg A -> #)
      -> (neg! A -> #).

?d : (pos A -> #)
      -> (pos? A -> #).
```

The obvious representation function $\ulcorner d \urcorner$ for proof terms can be extracted from this signature. As an example we give a derivation of $A \otimes B \longrightarrow B \otimes A$ and its representation in Elf.

$$\dfrac{\dfrac{\dfrac{}{B \longrightarrow B}I \quad \dfrac{}{A \longrightarrow A}I}{A, B \longrightarrow B \otimes A}\otimes R}{A \otimes B \longrightarrow B \otimes A}\otimes L$$

If we label the sequent $n{:}A \otimes B \xrightarrow{d} p{:}B \otimes A$ then the proof term d would be

$$\text{timesl}\,(\lambda n_1{:}A.\ \lambda n_2{:}B.\ \text{timesr}\,(\lambda p_2{:}B.\ \text{axiom}\,n_2\,p_2)\,(\lambda p_1{:}A.\ \text{axiom}\,n_1\,p_1)\,p)\,n.$$

Its representation in Elf is the object below. Note that we abstract over `A` and `B` to express that the derivation is parametric in `A` and `B`.

```
([A:o] [B:o]
   [n:neg (A times B)]
   [p:pos (B times A)]
   timesl ([n1:neg A] [n2:neg B]
             timesr ([p2:pos B] axiom n2 p2)
             ([p1:pos A] axiom n1 p1)
             p)
   n)
:
{A:o} {B:o}
   neg (A times B)
   -> pos (B times A)
   -> #.
```

The representation of the derivation

$$\dfrac{\dfrac{\dfrac{}{A \longrightarrow \top}\,\top R \qquad \dfrac{}{B \longrightarrow \top}\,\top R}{A, B \longrightarrow \top \otimes \top}\,\otimes R}{A \otimes B \longrightarrow \top \otimes \top}\,\otimes L$$

is the term

```
([A:o] [B:o]
   [n:neg (A times B)]
   [p:pos (top times top)]
   timesl ([n1:neg A] [n2:neg B]
             timesr ([p1:pos top] topr p1)
             ([p2:pos top] topr p2)
             p)
   n)
:
{A:o} {B:o}
   neg (A times B)
   -> pos (top times top)
   -> #.
```

which is ambiguous. For example, it is also the proof term for

$$\dfrac{\dfrac{\dfrac{}{A, B \longrightarrow \top}\,\top R \qquad \dfrac{}{\cdot \longrightarrow \top}\,\top R}{A, B \longrightarrow \top \otimes \top}\,\otimes R}{A \otimes B \longrightarrow \top \otimes \top}\,\otimes L$$

This ambiguity is resolved in the check of linearity constraints discussed below.

5.3 Linearity Constraints

Third, the representation of the linearity constraints. We need to check that certain meta-level functions occurring in proof terms are linear in their argument. Due to typing considerations and the absence of polymorphism in LF, we need two judgments to represent linearity:

```
linp : (pos A -> #) -> type.
linn : (neg A -> #) -> type.
```

We also require a judgment enforcing that each function occurring embedded in a proof term satisfies the appropriate linearity constraints.

```
lin  : # -> type.
```

We show here only parts of the implementations of these type families; the complete code may be found in Appendix A.3. Because we intend the type families to be used operationally in the Elf meta-language to determine if a given proof term representation is indeed linearly valid, we use notation `A <- B <- C` for `C -> B -> A`. The operational interpretation reads: "To solve a goal of the form `A`, first solve `B` and then `C`". First, some cases for `linp`.

Axiom. An axiom is linear in the positive formula participating in it.

```
linp_axiom : linp ([p] axiom N p).
```

Tensor. An application of $\otimes R$ is linear in a positive formula $p{:}A$ if (1) p labels the principal formula of the inference and p does not occur in either premise,

```
linp_timesr_0 : linp ([p] timesr D1 D2 p).
```

(2) p labels a side formula of the inference and p occurs linearly in the left premise and not at all in the right premise,

```
linp_timesr_1 : linp ([p] timesr (D1 p) D2 P)
                <- ({p1} linp ([p] D1 p p1)).
```

(3) p labels a side formula of the inference and p occurs linearly in the right premise and not at all the left premise

```
linp_timesr_2 : linp ([p] timesr D1 (D2 p) P)
                <- ({p2} linp ([p] D2 p p2)).
```

Note how the multiplicative nature of the tensor is encoded in these three inference rules. Note also how the variable hygiene of the meta-language captures occurrence conditions. The (implicit) quantifiers for `D1`, `D2`, and `P` are on the outside and thus, for example, in the last rule `D1` may not depend on `p`, while `D2` may, since we permit dependence by writing `(D2 p)`. Similarly, the principal formula `P` cannot be `p`. The left rule for tensor is simpler: p cannot label the principal formula, since p is positive and the principal formula is negative. We therefore have only one case.

```
linp_timesl   : linp ([p] timesl (D1 p) N)
                <- ({n1} {n2} linp ([p] D1 p n1 n2)).
```

Multiplicative Unit. This behaves like a 0-ary tensor.

```
linp_oner_0   : linp ([p] oner p).

linp_onel     : linp ([p] onel (D1 p) N)
                <- linp ([p] D1 p).
```

Additive Conjunction. An application of $\& R$ is linear in a positive formula $p{:}A$ if (1) p labels the principal formula of the inference and p does not occur in either premise,

```
linp_andr_0   : linp ([p] andr D1 D2 p).
```

(2) p labels a side formula of the inference and p occurs linearly in *both* premises

```
linp_andr     : linp ([p] andr (D1 p) (D2 p) P)
                 <- ({p1} linp ([p] D1 p p1))
                 <- ({p2} linp ([p] D2 p p2)).
```

This encodes the additive nature of the & connective. There are two left rules for &. In each case, p cannot be the principal formula (which is negative) and must therefore occur linearly in the premise:

```
linp_andl1    : linp ([p] andl1 (D1 p) N)
                 <- ({n1} linp ([p] D1 p n1)).

linp_andl2    : linp ([p] andl2 (D2 p) N)
                 <- ({n2} linp ([p] D2 p n2)).
```

Additive Unit. The additive unit $\top$ behaves like a 0-ary additive conjunction.

```
linp_topr_0   : linp ([p] topr p).

linp_topr     : linp ([p] topr P).
```

Involution. The involutive negation is straightforward.

```
linp_perpr_0  : linp ([p] perpr D1 p).

linp_perpr    : linp ([p] perpr (D1 p) P)
                 <- ({n1} linp ([p] D1 p n1)).

linp_perpl    : linp ([p] perpl (D1 p) N)
                 <- ({p1} linp ([p] D1 p p1)).
```

Universal Quantification. The universal quantifier does not introduce any new ideas: a formula occurrence is either principal or occurs linearly in the premise.

```
linp_forallr_0 : linp ([p] forallr D1 p).

linp_forallr  : linp ([p] forallr (D1 p) P)
                 <- ({a} {p1} linp ([p] D1 p a p1)).

linp_foralll  : linp ([p] foralll T (D1 p) N)
                 <- ({n1} linp ([p] D1 p n1)).
```

Exponentials. The modal operator ! introduces some new considerations, since we must mediate between modal and linear hypotheses and conclusions. We repeat the inference rules.

$$\frac{\Psi; \cdot \xrightarrow{d} p_1{:}A; \Theta}{\Psi; \cdot \xrightarrow{!\mathrm{r}\cdot(\lambda p_1:A.\ d)\ p} p{:}!A; \Theta}\,!R \qquad \frac{(\Psi, n^\omega{:}A); \Gamma \xrightarrow{d} \Delta; \Theta}{\Psi; (\Gamma, n{:}!A) \xrightarrow{!\mathrm{l}\ (\lambda^\omega n^\omega:A.\ d)\ n} \Delta; \Theta}\,!L$$

The $!R$ rule is constrained to have empty linear zones, that is, no linear side formulas are permitted. Thus an deduction ending in $!R$ is linear in a positive formula p only if p labels the principal formula of the inference (it cannot occur as a side formula).

```
linp_!r_0     : linp ([p] !r D1 p).
```

In the left rule we have no such restriction. Since p is positive, it must label a side formula.

```
linp_!l       : linp ([p] !l (D1! p) N)
                 <- ({n1!} linp ([p] D1! p n1!)).
```

Note that `n1!`, the newly introduced negative formula, falls into the modal zone, but this becomes important only later when we check linearity of subterms where required. For similar reasons, the structural rule for the exponential ! is simple.

```
linp_!d       : linp ([p] !d (D1 p) N!)
                 <- ({n1} linp ([p] D1 p n1)).
```

Analogous considerations for the ? operator lead to the following three rules.

```
linp_?r_0     : linp ([p] ?r D1 p).

linp_?r       : linp ([p] ?r (D1? p) P)
                 <- ({p1?} linp ([p] D1? p p1?)).

% no linp_?l_0: p is positive
% no linp_?l: p may not occur in D1

linp_?d       : linp ([p] ?d (D1 p) P?)
                 <- ({p1} linp ([p] D1 p p1)).
```

Note that p may not be the principal formula of a dereliction, which must originate from the positive modal zone.

We skip the dual rules for linearity of functions in negative formulas. We still have to define the type family `lin`, encoding the constraint that every linear hypothesis or conclusion that is introduced into a derivation is indeed used linearly. There is exactly one declaration for `lin` for each inference rule.

Axiom. There are no subderivations, so any use of an axiom is considered linear. The constraint that there be no linear side formulas is enforced collectively by checking all hypotheses and conclusions wherever they are introduced.

```
lin_axiom  : lin (axiom N P).
```

Tensor. Recall the right rule:

$$\frac{\Psi; \Gamma_1 \xrightarrow{d_1} p_1{:}A, \Delta_1; \Theta \qquad \Psi; \Gamma_2 \xrightarrow{d_2} p_2{:}B, \Delta_2; \Theta}{\Psi; \Gamma_1, \Gamma_2 \xrightarrow{\mathrm{timesr}\,(\lambda p_1{:}A.\ d_1)\,(\lambda p_2{:}B.\ d_2)\,p} p{:}A \otimes B, \Delta_1, \Delta_2; \Theta}\otimes R$$

We have to check that (1) the derivation d_1 of the left premise is linear in p_1, (2) the derivation d_2 of the right premise is linear in p_2, (3) all subderivations of the left premise are linear, and (4) all subderivations of the right premise are linear. That is:

```
lin_timesr : lin (timesr D1 D2 P)
              <- linp D1
              <- linp D2
              <- ({p1} lin (D1 p1))
              <- ({p2} lin (D2 p2)).
```

Recall the left rule:

$$\frac{\Psi; \Gamma, n_1{:}A, n_2{:}B \xrightarrow{d_1} \Delta; \Theta}{\Psi; \Gamma, n{:}A \otimes B \overset{\text{timesl}\,(\lambda n_1{:}A.\ \lambda n_2{:}B.\ d_1)\,n}{\longrightarrow} \Delta; \Theta} \otimes L$$

We have to check that (1) the derivation d_1 is linear in n_1, (2) the derivation d_1 is linear in n_2, and (3) all subderivations of d_1 are linear.

```
lin_timesl : lin (timesl D1 N)
              <- ({n2} linn ([n1] D1 n1 n2))
              <- ({n1} linn ([n2] D1 n1 n2))
              <- ({n1} {n2} lin (D1 n1 n2)).
```

The other multiplicative and additive connectives and the quantifiers are congruences in a similar style. They are given in Appendix A.3.

Exponentials. Recall the left and right rules for the ! modality:

$$\frac{\Psi; \cdot \xrightarrow{d_1} p_1{:}A; \Theta}{\Psi; \cdot \overset{!\text{r}\cdot(\lambda p_1{:}A.\ d_1)\,p}{\longrightarrow} p{:}!A; \Theta} !R \qquad \frac{(\Psi, n^\omega{:}A); \Gamma \xrightarrow{d_1} \Delta; \Theta}{\Psi; (\Gamma, n{:}!A) \overset{!\text{l}\,(\lambda^\omega n^\omega{:}A.\ d_1)\,n}{\longrightarrow} \Delta; \Theta} !L$$

To check that an application of $!R$ is linear, we need to check (1) that the derivation d_1 of the premise is linear in p_1, and (2) that all subderivations of the premise are linear. The condition that there may be no linear side formulas is encoded in the `linp` and `linn` families.

```
lin_!r     : lin (!r D1 P)
              <- linp D1
              <- ({p1} lin (D1 p1)).
```

In the $!L$ rule, the newly introduced hypothesis $n^\omega{:}A$ is modal and therefore does not need to satisfy any linearity constraints. For example, it may occur as a side formula to other applications of $!R$. That is, we only need to check that all subderivations of the premise are again linear.

```
lin_!l     : lin (!l D1! N)
              <- ({n1!} lin (D1! n1!)).
```

Dereliction introduces a new *linear* hypothesis which must thus be checked, together with the linearity of all further subderivations.

```
lin_!d     : lin (!d D1 N!)
              <- linn D1
              <- ({n1} lin (D1 n1)).
```

The cases for the ? modality are dual.

```
lin_?r      : lin (?r D1? P)
                <- ({p1?} lin (D1? p1?)).

lin_?l      : lin (?l D1 N)
                <- linn D1
                <- ({n1} lin (D1 n1)).

lin_?d      : lin (?d D1 P?)
                <- linp D1
                <- ({p1} lin (D1 p1)).
```

At the top-level we must check that the derivation is linear in each linear hypothesis or conclusion, and also that every subderivation is linear. A common case is one where we have one linear hypothesis and one linear conclusion, represented by the type `neg A -> pos B -> #`. To check a proof term for validity we may use the following auxiliary judgment.

```
lin2 : (neg B -> pos C -> #) -> type.

lin2_all : lin2 D
                <- ({p} linn ([n] D n p))  % linear in n
                <- ({n} linp ([p] D n p))  % linear in p
                <- ({n} {p} lin (D n p)).  % subderivations are all linear
```

The adequacy theorem for our representation is tedious, but we include some details since it may be instructive. We fix the signature Σ to the one consisting of all declarations summarized in Appendices A.1–A.3 (after type reconstruction) and write $G \vdash^{LF} M \Uparrow A$ if M is a canonical LF object of type A in context G. Assume we have sequent derivation ending in

$$\Psi; \Gamma \xrightarrow{d} \Delta; \Theta.$$

For each hypothesis or conclusion we introduce a typing assumption in LF, where the zones are distinguished by four type families, `neg!`, `neg`, `pos`, and `pos?`.

$$\begin{array}{rclcl}
\ulcorner\Psi\urcorner & = & \ulcorner n_1^\omega{:}A_1,\ldots,n_j^\omega{:}A_j\urcorner & = & n_1^\omega{:}\mathtt{neg!}\ulcorner A_1\urcorner,\ldots,n_j^\omega{:}\mathtt{neg!}\ulcorner A_j\urcorner \\
\ulcorner\Gamma\urcorner & = & \ulcorner n_1{:}B_1,\ldots,n_k{:}B_k\urcorner & = & n_1{:}\mathtt{neg}\ulcorner B_1\urcorner,\ldots,n_k{:}\mathtt{neg}\ulcorner B_k\urcorner \\
\ulcorner\Delta\urcorner & = & \ulcorner p_1{:}C_1,\ldots,p_l{:}C_l\urcorner & = & p_1{:}\mathtt{pos}\ulcorner C_1\urcorner,\ldots,p_l{:}\mathtt{pos}\ulcorner C_l\urcorner \\
\ulcorner\Theta\urcorner & = & \ulcorner p_1^\omega{:}D_1,\ldots,p_m^\omega{:}D_m\urcorner & = & p_1^\omega{:}\mathtt{pos?}\ulcorner D_1\urcorner,\ldots,p_m^\omega{:}\mathtt{pos?}\ulcorner D_m\urcorner
\end{array}$$

We ambiguously apply $\ulcorner\cdot\urcorner$ to Ψ, Γ, Δ, and Θ, since it is always clear in which zone the hypotheses or conclusions originate. We then have the theorem that the representation of every derivation is well-typed and canonical in the LF context which arises from translating the hypotheses and conclusions as defined above.

Lemma 4 (Soundness of Representation) *Let $\mathcal{D} :: (\Psi; \Gamma \xrightarrow{d} \Delta; \Theta)$ be a sequent derivation with parameters among $a_1, \ldots, a_h$. Then*

1. $a_1{:}\mathtt{i}, \ldots, a_h{:}\mathtt{i}, \ulcorner\Psi\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner, \ulcorner\Theta\urcorner \vdash^{LF} \ulcorner d\urcorner \Uparrow \mathtt{\#}$,

2. *there exists an M such that*

$$a_1{:}\mathtt{i}, \ldots, a_h{:}\mathtt{i}, \ulcorner\Psi\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner, \ulcorner\Theta\urcorner \vdash^{LF} M \Uparrow \mathtt{lin}\,\ulcorner d \urcorner$$

3. *for each $n_i{:}\mathrm{neg}\ulcorner B_i \urcorner$, $1 \leq i \leq k$, there exists an N_i such that*

$$a_1{:}\mathtt{i}, \ldots, a_h{:}\mathtt{i}, \ulcorner\Psi\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner, \ulcorner\Theta\urcorner \vdash^{LF} N_i \Uparrow \mathtt{linn}\,(\lambda n_i{:}\mathtt{neg}\ulcorner B_i \urcorner.\ \ulcorner d \urcorner)$$

4. *for each $p_i{:}\mathrm{pos}\ulcorner C_i \urcorner$, $1 \leq i \leq l$ in $\ulcorner\Delta\urcorner$ there exists a P_i such that*

$$a_1{:}\mathtt{i}, \ldots, a_h{:}\mathtt{i}, \ulcorner\Psi\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner, \ulcorner\Theta\urcorner \vdash^{LF} P_i \Uparrow \mathtt{linp}\,(\lambda p_i{:}\mathtt{pos}\,\ulcorner C_i \urcorner.\ \ulcorner d \urcorner)$$

Proof: By induction on the structure of $\mathcal{D}$. □

In the other direction (the more difficult one) we have to show that every well-typed, canonical LF object satisfying conditions (1)–(4) above is the representation of a linear sequent derivation.

Lemma 5 (Completeness of Representation) *Let G be an LF context of the form*

$$a_1{:}\mathtt{i}, \ldots, a_h{:}\mathtt{i}, \ulcorner\Psi\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner, \ulcorner\Theta\urcorner$$

as defined above. If

1. $G \vdash^{LF} D \Uparrow \mathtt{\#}$,
2. $G \vdash^{LF} M \Uparrow \mathtt{lin}\,D$,
3. $G \vdash^{LF} N_i \Uparrow \mathtt{linn}\,(\lambda n_i{:}\mathtt{neg}\ulcorner B_i \urcorner.\ D)$ *for* $1 \leq i \leq k$, *and*
4. $G \vdash^{LF} P_i \Uparrow \mathtt{linp}\,(\lambda p_i{:}\mathtt{pos}\ulcorner C_i \urcorner.\ D)$ *for* $1 \leq i \leq l$,

then there exist a proof term d and a derivation $\mathcal{D} :: (\Psi; \Gamma \xrightarrow{d} \Delta; \Theta)$ such that $\ulcorner d \urcorner = D$.

Proof: By induction on the structure of the derivation $\mathcal{L} :: (G \vdash^{LF} D \Uparrow \mathtt{\#})$, applying inversion on the derivations postulated by items (2)–(4). It is crucial that D, M, N_i and P_i be in canonical form. □

Soundness and completeness are the crucial lemmas for adequacy of the representation, since it can be easily checked that it is a bijection. Compositionality is guaranteed by construction since we represented object-level variables by meta-level variables.

Theorem 6 (Adequacy of Representation) *There exists a bijection $\ulcorner\cdot\urcorner$ between linear sequent derivations of*

$$\begin{array}{c} \mathcal{D} \\ \Psi; \Gamma \xrightarrow{d} \Delta; \Theta \end{array}$$

in the system LV and canonical LF objects of type $\mathtt{lin}\ulcorner d \urcorner$ satisfying conditions (2)–(4) in an appropriate context as stated in Lemmas 4 and 5. The bijection is compositional in the sense that

$$\ulcorner [t/a]\mathcal{D} \urcorner = [\ulcorner t \urcorner/a]\ulcorner \mathcal{D} \urcorner \quad \textit{and} \quad \ulcorner [n_1/n_2]\mathcal{D} \urcorner = [n_1/n_2]\ulcorner \mathcal{D} \urcorner \quad \textit{and} \quad \ulcorner [p_1/p_2]\mathcal{D} \urcorner = [p_1/p_2]\ulcorner \mathcal{D} \urcorner$$

While this is a very complex statement, it is actually quite easy to use. The complexity is mostly due to the fact that we have to admit an undetermined number of hypothesis and conclusions in its formulation. We can specialize it to the case where we have no linear hypotheses or conclusions (at the end sequent), or to the case where we have precisely one linear hypotheses and conclusion. For example, if we declare

```
lin2 : (neg B -> pos C -> #) -> type.

lin2_all : lin2 D
              <- ({p} linn ([n] D n p))  % linear in n
              <- ({n} linp ([p] D n p))  % linear in p
              <- ({n} {p} lin (D n p)).  % subderivations are all linear
```

then we can easily show, using Theorem 6:

Corollary 7 *There is a compositional bijection between linear sequent derivations of*

$$\begin{array}{c}\mathcal{D}\\ \Psi; B \xrightarrow{d} C; \Theta\end{array}$$

with parameters among $a_1, \ldots, a_h$ in the system LV and canonical LF objects M such that

$$a_1{:}\mathtt{i}, \ldots, a_h{:}\mathtt{i}, \ulcorner\Psi\urcorner, \ulcorner\Theta\urcorner \vdash^{LF} M \Uparrow \mathtt{lin2}\,(\lambda n{:}\mathtt{neg}\ulcorner B\urcorner.\ \lambda p{:}\mathtt{pos}\ulcorner C\urcorner.\ \ulcorner d\urcorner).$$

We now return to an earlier example, checking that the given proof term actually represents a derivation of $A \otimes B \longrightarrow B \otimes A$. To this end we check via an Elf query that it satisfies `lin2`. Elf here is used as a constraint logic programming language, searching for a closed instance of the query. This search is incomplete (depth-first, left-to-right subgoal selection), but `linn`, `linp`, and `lin` are all operationally adequate and not merely a declarative description of the concepts of linearity. The answer substitution for `L` below can be considered the representation of the sequent derivation according to Corollary 7.

```
?- L :
{A:o} {B:o}
lin2
([n:neg (A times B)]
   [p:pos (B times A)]
   timesl ([n1:neg A] [n2:neg B]
             timesr ([p2:pos B] axiom n2 p2)
             ([p1:pos A] axiom n1 p1)
             p)
   n).
Solving...

L =
   [A:o] [B:o]
      lin2_all
         ([n:neg (A times B)] [p:pos (B times A)]
             lin_timesl
                ([n1:neg A] [n2:neg B]
```

```
              lin_timesr ([p2:pos A] lin_axiom) ([p1:pos B] lin_axiom)
                 linp_axiom linp_axiom)
          ([n1:neg A] linn_timesr_1 [p1:pos B] linn_axiom)
          ([n2:neg B] linn_timesr_2 [p2:pos A] linn_axiom))
     ([n:neg (A times B)] linp_timesl [n1:neg A] [n2:neg B] linp_timesr_0)
     ([p:pos (B times A)] linn_timesl_0).
```

The following is an *incorrect* derivation: It violates the multiplicative nature of the tensor and the side conditions on axioms.

$$\dfrac{\dfrac{\dfrac{}{A, B \longrightarrow B}\,I}{A \otimes B \longrightarrow B}\otimes L \qquad \dfrac{\dfrac{}{A, B \longrightarrow A}\,I}{A \otimes B \longrightarrow A}\otimes L}{A \otimes B \longrightarrow B \otimes A}\otimes R$$

A proof term for this derivation may still be written down:

```
([A:o] [B:o]
   [n:neg (A times B)]
   [p:pos (B times A)]
   timesr ([p2:pos B] timesl ([n1:neg A] [n2:neg B] axiom n2 p2) n)
   ([p1:pos A] timesl ([n1:neg A] [n2:neg B] axiom n1 p1) n)
   p)
:
{A:o} {B:o}
   neg (A times B)
   -> pos (B times A)
   -> #.
```

However, this term does not satisfy the linearity constraints and thus does not represent a derivation.

```
?- {A:o} {B:o}
lin2
([n:neg (A times B)]
   [p:pos (B times A)]
   timesr ([p2:pos B] timesl ([n1:neg A] [n2:neg B] axiom n2 p2) n)
   ([p1:pos A] timesl ([n1:neg A] [n2:neg B] axiom n1 p1) n)
   p).
Solving...
no
```

The same proof term may also annotate several valid derivations. This phenomenon was illustrated in the examples toward the end of the last section. We show here two of the possible four derivations with the same proof term.

$$\dfrac{\dfrac{\dfrac{}{A \longrightarrow \top}\,\top R \qquad \dfrac{}{B \longrightarrow \top}\,\top R}{A, B \longrightarrow \top \otimes \top}\otimes R}{A \otimes B \longrightarrow \top \otimes \top}\otimes L \qquad\qquad \dfrac{\dfrac{\dfrac{}{A, B \longrightarrow \top}\,\top R \qquad \dfrac{}{\cdot \longrightarrow \top}\,\top R}{A, B \longrightarrow \top \otimes \top}\otimes R}{A \otimes B \longrightarrow \top \otimes \top}\otimes L$$

The implementation of the linearity constraints enumerates all possible derivations whose annotation is the given proof term. This always terminates, since there are only finitely many possibilities.

The order of enumeration depends on the order of the overlapping clauses for `linn` and `linp` since they are tried first-to-last.

```
?- L :
     {A:o} {B:o}
     lin2
     [n:neg (A times B)]
     [p:pos (top times top)]
     timesl ([n1:neg A] [n2:neg B]
               timesr ([p1:pos top] topr p1)
               ([p2:pos top] topr p2)
               p)
     n.
Solving...

L =
   [A:o] [B:o]
      lin2_all
         ([n:neg (A times B)] [p:pos (top times top)]
             lin_timesl
                ([n1:neg A] [n2:neg B]
                    lin_timesr ([p2:pos top] lin_topr) ([p1:pos top] lin_topr)
                       linp_topr_0 linp_topr_0)
                ([n1:neg A] linn_timesr_1 [p1:pos top] linn_topr)
                ([n2:neg B] linn_timesr_1 [p1:pos top] linn_topr))
         ([n:neg (A times B)] linp_timesl [n1:neg A] [n2:neg B] linp_timesr_0)
         ([p:pos (top times top)] linn_timesl_0).
;

L =
   [A:o] [B:o]
      lin2_all
         ([n:neg (A times B)] [p:pos (top times top)]
             lin_timesl
                ([n1:neg A] [n2:neg B]
                    lin_timesr ([p2:pos top] lin_topr) ([p1:pos top] lin_topr)
                       linp_topr_0 linp_topr_0)
                ([n1:neg A] linn_timesr_2 [p2:pos top] linn_topr)
                ([n2:neg B] linn_timesr_1 [p1:pos top] linn_topr))
         ([n:neg (A times B)] linp_timesl [n1:neg A] [n2:neg B] linp_timesr_0)
         ([p:pos (top times top)] linn_timesl_0).
;

L =
   [A:o] [B:o]
      lin2_all
         ([n:neg (A times B)] [p:pos (top times top)]
             lin_timesl
                ([n1:neg A] [n2:neg B]
                    lin_timesr ([p2:pos top] lin_topr) ([p1:pos top] lin_topr)
                       linp_topr_0 linp_topr_0)
                ([n1:neg A] linn_timesr_1 [p1:pos top] linn_topr)
```

```
               ([n2:neg B] linn_timesr_2 [p2:pos top] linn_topr))
         ([n:neg (A times B)] linp_timesl [n1:neg A] [n2:neg B] linp_timesr_0)
         ([p:pos (top times top)] linn_timesl_0).
;

L =
   [A:o] [B:o]
      lin2_all
         ([n:neg (A times B)] [p:pos (top times top)]
             lin_timesl
                ([n1:neg A] [n2:neg B]
                    lin_timesr ([p2:pos top] lin_topr) ([p1:pos top] lin_topr)
                       linp_topr_0 linp_topr_0)
                ([n1:neg A] linn_timesr_2 [p2:pos top] linn_topr)
                ([n2:neg B] linn_timesr_2 [p2:pos top] linn_topr))
         ([n:neg (A times B)] linp_timesl [n1:neg A] [n2:neg B] linp_timesr_0)
         ([p:pos (top times top)] linn_timesl_0).
;
no more solutions
```

6 Admissibility of Cut

In this section we prove the admissibility of cut in LV. The admissibility of cut in CLL is a simple corollary by virtue of the translations between LV and CLL from Theorem 2.

Theorem 8 (Admissibility of Cut in LV) *The three rules of cut*

$$\frac{\Psi; \Gamma_1 \longrightarrow A, \Delta_1; \Theta \qquad \Psi; \Gamma_2, A \longrightarrow \Delta_2; \Theta}{\Psi; \Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2; \Theta}\,\mathrm{Cut}$$

$$\frac{\Psi; \cdot \longrightarrow A; \Theta \qquad (\Psi, A); \Gamma \longrightarrow \Delta; \Theta}{\Psi; \Gamma \longrightarrow \Delta; \Theta}\,\mathrm{Cut!} \qquad \frac{\Psi; \Gamma \longrightarrow \Delta; (A, \Theta) \qquad \Psi; A \longrightarrow \cdot; \Theta}{\Psi; \Gamma \longrightarrow \Delta; \Theta}\,\mathrm{Cut?}$$

for the linear sequent calculus LV *are admissible.*

Proof: Let A be the cut formula and d and e the proof terms of the derivations of the premises of the cut rules. We proceed by three nested structural inductions on A, d and e, simultaneously for Cut, Cut!, and Cut?. Appeals to Cut! and Cut? are considered greater than Cut, if the cut formula A is the same. To rephrase: We may appeal to the induction hypothesis on (1) a smaller cut formula, (2) the same cut formula, but pass from Cut! or Cut? to Cut, (3) the same cut formula and rule, but smaller proof term d, or (4) the same cut formula, rule, proof terms d, but smaller proof term e. Actually, there is no natural priority between d and e: Whenever we need the induction hypothesis for a smaller e, d remains the same, and vice versa.

In the proof we distinguish the following classes of cases.

1. Axiom cases. Either $\mathcal{D}$ or $\mathcal{E}$ is an axiom, with the cut formula as principal formula.

2. Essential cases. The cut formula is the principal formula of the last inference in both $\mathcal{D}$ and $\mathcal{E}$.

3. Structural cases. The cut formula is the principal formula of a structural rule ($!D$ or $?D$) at the end of $\mathcal{D}$ or $\mathcal{E}$. There are only two such cases.

4. Commutative cases. The cut formula is a side formula of the last inference in $\mathcal{D}$ or $\mathcal{E}$. There are three subclasses of this class, depending on whether we are considering Cut, Cut! or Cut?.

These cover all possible cases of Cut, Cut! and Cut?. The general idea of the proof in each class of cases is summarized below.

1. Axiom cases. Here the cut can be eliminated outright.

2. Essential cases. Here we can eliminate the cut by appeals to the induction hypothesis with immediate subformulas of the cut formula A.

3. Structural cases. Here we appeal to the induction hypotheses on an immediate subderivation followed by a switch from Cut! or Cut? to Cut, maintaining the same cut formula A.

4. Commutative cases. Here we appeal to the induction hypothesis on the proof terms labeling each immediate subderivation of $\mathcal{D}$ or $\mathcal{E}$. It is also in some of these cases that we need weakening for the non-linear zones without changing the proof term.

Many of the cases are very similar. For each class we only show some representative cases below.

The algorithm for admissibility of cut that may be extracted from our proof is non-deterministic, since the various cases are not mutually exclusive. Our implementation computes a proof term for the conclusion, given proof terms for the premises of a cut. In Elf, this is represented as a *relation* between A, d, e (the input arguments) and f (the resulting derivation). Since there are three forms of cut, we have three corresponding type families.

```
ad  : {A:o} (pos A -> #) -> (neg A -> #) -> # -> type.
ad! : {A:o} (pos A -> #) -> (neg! A -> #) -> # -> type.
ad? : {A:o} (pos? A -> #) -> (neg A -> #) -> # -> type.
```

Reasoning about linearity constraints or distribution of hypotheses is not implemented in these type families. We therefore do not consider them an implementation of the proof, but only of some of its operational aspects. Nonetheless, it is sufficient to non-deterministically calculate proof terms for the conclusion of a cut. The complete implementation is given in Appendix A.4. An example of its execution may be found in Appendix A.5.

Axiom Cases. These comprise two cases: either $\mathcal{D}$ or $\mathcal{E}$ is an axiom with the principal formula being the cut formula. Since axioms permit no linear side formulas only two further commutative cases (for Cut! and Cut?) involving axioms arise. We show the case where $\mathcal{D}$ is an axiom; the other case is symmetric.

Case:

$$\mathcal{D} = \cfrac{}{\Psi; N{:}A \overset{\text{axiom}\,N\,p}{\longrightarrow} p{:}A; \Theta}\, I$$

and $\mathcal{E} :: (\Psi; \Gamma_2, n{:}A \xrightarrow{e} \Delta_2; \Theta)$ is arbitrary. Then we construct

$$\mathcal{F} = \begin{array}{c} [N/n]\mathcal{E} \\ \Psi; \Gamma_2, N{:}A \overset{[N/n]e}{\longrightarrow} \Delta_2; \Theta \end{array}$$

```
ad_axiom_l : ad A ([p] axiom N p) E (E N).
```

In the remaining cases we omit the proof terms, since they may easily be reconstructed from the derivations and clutter the presentation. In cases we proof terms are important for the induction we assign proof term e to derivation $\mathcal{E}$, d to $\mathcal{D}$, d_1 to $\mathcal{D}_1$, etc.

Essential Cases. These are the cases where the cut formula A is the principal formula of the last inference in $\mathcal{D}$ and $\mathcal{E}$.

Case:

$$\mathcal{D} = \cfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Psi; \Gamma_1' \longrightarrow A_1, \Delta_1'; \Theta & \Psi; \Gamma_1'' \longrightarrow A_2, \Delta_1''; \Theta \end{array}}{\Psi; \Gamma_1', \Gamma_1'' \longrightarrow A_1 \otimes A_2, \Delta_1', \Delta_1''; \Theta} \otimes R$$

and

$$\mathcal{E} = \cfrac{\begin{array}{c} \mathcal{E}_1 \\ \Psi; \Gamma_2, A_1, A_2 \longrightarrow \Delta_2; \Theta \end{array}}{\Psi; \Gamma_2, A_1 \otimes A_2 \longrightarrow \Delta_2; \Theta} \otimes L.$$

Then we construct

$\mathcal{E}_1' :: (\Psi; \Gamma_1', \Gamma_2, A_2 \longrightarrow \Delta_1', \Delta_2; \Theta)$	By ind. hyp. on A_1 from $\mathcal{D}_1$ and $\mathcal{E}_1$
$\mathcal{F} :: (\Psi; \Gamma_1', \Gamma_1'', \Gamma_2, \longrightarrow \Delta_1', \Delta_1'', \Delta_2; \Theta)$	By ind. hyp. on A_2 from $\mathcal{D}_2$ and $\mathcal{E}_1'$

```
ad_times : ad (A1 times A2) ([p] timesr D1 D2 p) ([n] timesl E1 n) F
            <- ({n2:neg A2} ad A1 D1 ([n1] E1 n1 n2) (E1' n2))
            <- ad A2 D2 E1' F.
```

In the remaining cases we omit "$\Psi;$" and "$;\Theta$" when they do not change throughout the argument as above. A full description of the case can be obtained by adjoining them to the left and right of every sequent in its abbreviated form.

Case:

$$\mathcal{D} = \cfrac{}{\cdot \longrightarrow 1} 1R$$

and

$$\mathcal{E} = \cfrac{\begin{array}{c} \mathcal{E}_1 \\ \Gamma \longrightarrow \Delta \end{array}}{\Gamma, 1 \longrightarrow \Delta} 1L.$$

Then $\mathcal{F} = \mathcal{E}_1$ is a derivation with the desired end sequent.

```
ad_one   : ad (one) ([p] oner p) ([n] onel E1 n) E1.
```

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma_1 \longrightarrow A_1, \Delta_1 & \Gamma_1 \longrightarrow A_2, \Delta_1 \end{array}}{\Gamma_1 \longrightarrow A_1 \& A_2, \Delta_1} \& R$$

and

$$\mathcal{E} = \dfrac{\begin{array}{c} \mathcal{E}_1 \\ \Gamma_2, A_1 \longrightarrow \Delta_2 \end{array}}{\Gamma_2, A_1 \& A_2 \longrightarrow \Delta_2} \& R_1$$

Then

$\mathcal{F} :: (\Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2)$ By ind. hyp. on A_1 from $\mathcal{D}_1$ and $\mathcal{E}_1$

```
ad_and1 : ad (A1 and A2) ([p] andr D1 D2 p) ([n] andl1 E1 n) F
           <- ad A1 D1 E1 F.
```

Case: The case where $\mathcal{E}$ ends in $\& R_2$ is symmetric to the previous one.

```
ad_and2 : ad (A1 and A2) ([p] andr D1 D2 p) ([n] andl2 E2 n) F
           <- ad A2 D2 E2 F.
```

Case: There is no essential case for the additive unit $\top$, since there is no left rule.

Case:

$$\mathcal{D} = \dfrac{\begin{array}{c} \mathcal{D}_1 \\ \Gamma_1, A_1 \longrightarrow \Delta_1 \end{array}}{\Gamma_1 \longrightarrow A_1^{\perp}, \Delta_1} \neg R$$

and

$$\mathcal{E} = \dfrac{\begin{array}{c} \mathcal{E}_1 \\ \Gamma_2 \longrightarrow A_1, \Delta_2 \end{array}}{\Gamma_2, A_1^{\perp} \longrightarrow \Delta_2} \neg L.$$

Then

$\mathcal{F} :: (\Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2)$ By ind. hyp. on A_1 from $\mathcal{E}_1$ and $\mathcal{D}_1$.

```
ad_perp : ad (perp A1) ([p] perpr D1 p) ([n] perpl E1 n) F
           <- ad A1 E1 D1 F.
```

Case:

$$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma_1 \longrightarrow [a/x]A_1, \Delta_1\end{matrix}}{\Gamma_1 \longrightarrow \forall x.\ A_1, \Delta_1}\forall R^a$$

and

$$\mathcal{E} = \dfrac{\begin{matrix}\mathcal{E}_1 \\ \Gamma_2, [t/x]A_1 \longrightarrow \Delta_1\end{matrix}}{\Gamma_2, \forall x.\ A_1 \longrightarrow \Delta_2}\forall L$$

Then

$[t/a]\mathcal{D}_1 :: (\Gamma_1 \longrightarrow [t/x]A_1, \Delta_1)$ Since $\mathcal{D}_1$ is parametric in a
$\mathcal{F} :: (\Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2)$ By ind. hyp. on $[t/x]A_1$ from $[t/a]\mathcal{D}_1$ and $\mathcal{E}_1$

Note that in first-order logic $[t/x]A_1$ is structurally smaller than $\forall x.\ A_1$; this would break down in an attempt to extend this proof directly to higher-order logic.

```
ad_forall : ad (forall A1) ([p] forallr D1 p) ([n] foralll T E1 n) F
             <- ad (A1 T) (D1 T) E1 F.
```

Case:

$$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Psi; \cdot \longrightarrow A_1; \Theta\end{matrix}}{\Psi; \cdot \longrightarrow !A_1; \Theta}!R$$

and

$$\mathcal{E} = \dfrac{\begin{matrix}\mathcal{E}_1 \\ (\Psi, A_1); \Gamma_2 \longrightarrow \Delta_2; \Theta\end{matrix}}{\Psi; \Gamma_2, !A_1 \longrightarrow \Delta_2; \Theta}!L.$$

Then

$\mathcal{F} :: (\Psi; \Gamma_2 \longrightarrow \Delta_2; \Theta)$ By ind. hyp. on A_1 with Cut! from $\mathcal{D}_1$ and $\mathcal{E}_1$.

```
ad_! : ad (! A1) ([p] !r D1! p) ([n] !l E1 n) F
        <- ad! A1 D1! E1 F.
```

Case:

$$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Psi; \Gamma_1 \longrightarrow \Delta_1; (A_1; \Theta)\end{matrix}}{\Psi; \Gamma_1 \longrightarrow ?A_1, \Delta_1; \Theta}?R$$

and

$$\mathcal{E} = \dfrac{\begin{matrix}\mathcal{E}_1 \\ \Psi; A_1 \longrightarrow \cdot; \Theta\end{matrix}}{\Psi; ?A_1 \longrightarrow \cdot; \Theta}?L.$$

Then

$\mathcal{F} :: (\Psi; \Gamma_1 \longrightarrow \Delta_1; \Theta)$ By ind. hyp. on A_1 with Cut? from $\mathcal{D}_1$ and $\mathcal{E}_1$.

```
ad_? : ad (? A1) ([p] ?r D1 p) ([n] ?l E1! n) F
        <- ad? A1 D1 E1! F.
```

Structural Cases. There are only two structural rules, $!D$ and $?D$, which copy a formula from a non-linear to the appropriate linear zone. In case the copied formula A is the principal formula of a Cut! or Cut?, we first eliminate the remaining non-linear hypothesis or conclusion A and then appeal to Cut on the same formula A. This is permissable in our structural induction: Cut is considered smaller than Cut! and Cut? if the cut formula does not change.

Case: $\mathcal{D} :: (\Psi; \cdot \longrightarrow A; \Theta)$ is arbitrary and

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_1\\ (\Psi, A); \Gamma_2, A \longrightarrow \Delta_2; \Theta\end{array}}{(\Psi, A); \Gamma_2 \longrightarrow \Delta_2; \Theta}\,!D.$$

Then (writing e_1 for the proof term annotating $\mathcal{E}_1$ and d for the proof term of $\mathcal{D}$)

$\mathcal{E}_1' :: (\Psi; \Gamma_2, A \longrightarrow \Delta_2; \Theta)$ By ind. hyp. on A, d, and e_1 with Cut! from $\mathcal{D}$ and $\mathcal{E}_1$
$\mathcal{F}_1 :: (\Psi; \Gamma_2 \longrightarrow \Delta_2; \Theta)$ By ind. hyp. on A with Cut from $\mathcal{D}$ and $\mathcal{E}_1'$

```
ad!_d  : ad! A D! ([n!] !d (E1 n!) n!) F
           <- ({n1:neg A} ad! A D! ([n!] E1 n! n1) (E1' n1))
           <- ad A D! E1' F.
```

Case: The case of a Cut? where $\mathcal{D}$ ends in $?D$ and $\mathcal{E}$ is arbitrary is dual to the previous one.

```
ad?_r  : ad? A ([p?] ?d (D1 p?) p?) E! F
           <- ({p1:pos A} ad? A ([p?] D1 p? p1) E! (D1' p1))
           <- ad A D1' E! F.
```

Commutative Cases of Cut. Here we have a Cut where the cut formula is a side formula of the last inference in $\mathcal{D}$ (left commutative cases) or $\mathcal{E}$ (right commutative cases). These cases are not mutually exclusive, so the algorithm implicit in this proof is naturally non-deterministic. We only show a few cases, since they are very repetitive: We appeal to the induction hypothesis on a subderivation and then re-apply the last inference.

Case: $\mathcal{D} :: (\Gamma_1 \longrightarrow A, \Delta_1)$ is arbitrary and

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_1\\ \Gamma_2', A \longrightarrow B_1, \Delta_2'\end{array} \qquad \begin{array}{c}\mathcal{E}_2\\ \Gamma_2'' \longrightarrow B_2, \Delta_2''\end{array}}{\Gamma_2', \Gamma_2'', A \longrightarrow B_1 \otimes B_2, \Delta_2', \Delta_2''}\,\otimes R.$$

Then (writing e_1 for the proof term annotating $\mathcal{E}_1$)

$\mathcal{E}_1' :: (\Gamma_1, \Gamma_2' \longrightarrow B_1, \Delta_1, \Delta_2')$ By ind. hyp. on A, d, and e_1 from $\mathcal{D}$ and $\mathcal{E}_1$
$\mathcal{F} :: (\Gamma_1, \Gamma_2', \Gamma_2'' \longrightarrow B_1 \otimes B_2, \Delta_1, \Delta_2', \Delta_2'')$ By $\otimes R$ from $\mathcal{E}_1'$ and $\mathcal{E}_2$.

There is a symmetric case where A occurs in the hypotheses of $\mathcal{E}_2$. We show both cases in Elf.

```
adr_timesr1 : ad A D ([n] timesr (E1 n) E2 P) (timesr E1' E2 P)
               <- ({p1:pos B1} ad A D ([n] E1 n p1) (E1' p1)).

adr_timesr2 : ad A D ([n] timesr E1 (E2 n) P) (timesr E1 E2' P)
               <- ({p2:pos B2} ad A D ([n] E2 n p2) (E2' p2)).
```

Case: $\mathcal{D} :: (\Gamma_1 \longrightarrow A, \Delta_1)$ is arbitrary and

$$\mathcal{E} = \cfrac{\begin{matrix}\mathcal{E}_1 \\ \Gamma_2, A \longrightarrow B_1, \Delta_2'\end{matrix} \qquad \begin{matrix}\mathcal{E}_2 \\ \Gamma_2, A \longrightarrow B_2, \Delta_2'\end{matrix}}{\Gamma_2, A \longrightarrow B_1 \& B_2, \Delta_2'}\,\&R.$$

Then (writing e_1 and e_2 for the proof terms annotating $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively)

$\mathcal{E}_1' :: (\Gamma_1, \Gamma_2 \longrightarrow B_1, \Delta_1, \Delta_2')$ By ind. hyp. on A, d, and e_1 from $\mathcal{D}$ and $\mathcal{E}_1$
$\mathcal{E}_2' :: (\Gamma_1, \Gamma_2 \longrightarrow B_2, \Delta_1, \Delta_2')$ By ind. hyp. on A, d, and e_2 from $\mathcal{D}$ and $\mathcal{E}_2$
$\mathcal{F} :: (\Gamma_1, \Gamma_2 \longrightarrow B_1 \& B_2, \Delta_1, \Delta_2')$ By $\&R$ from $\mathcal{E}_1'$ and $\mathcal{E}_2'$.

```
adr_andr : ad A D ([n] andr (E1 n) (E2 n) P) (andr E1' E2' P)
            <- ({p1:pos B1} ad A D ([n] E1 n p1) (E1' p1))
            <- ({p2:pos B2} ad A D ([n] E2 n p2) (E2' p2)).
```

Case: $\mathcal{D} :: (\Gamma_1 \longrightarrow A, \Delta_1)$ is arbitrary and $\mathcal{E}$ ends in $!R$ with A as a side formula. This case is impossible, since $!R$ permits no linear side formulas.

Case: $\mathcal{D} :: (\Psi; \Gamma_1 \longrightarrow A, \Delta_1; \Theta)$ is arbitrary and

$$\mathcal{E} = \cfrac{\begin{matrix}\mathcal{E}_1 \\ (\Psi, B_1); \Gamma_2', A \longrightarrow \Delta_2; \Theta\end{matrix}}{\Psi; \Gamma_2', !B_1, A \longrightarrow \Delta_2; \Theta}\,!L$$

In this case we cannot directly appeal to the induction hypothesis, since the non-linear hypotheses of $\mathcal{E}_1$ contain B_1, but not those of $\mathcal{D}$. However, we can apply weakening (WL, Lemma 3) to $\mathcal{D}$ without changing its proof term d. Thus (writing e_1 for the proof term of $\mathcal{E}_1$)

$(\mathcal{D}, B_1) :: ((\Psi, B_1); \Gamma_1 \xrightarrow{d} A, \Delta_1; \Theta)$ By Lemma 3
$\mathcal{E}_1' :: ((\Psi, B_1); \Gamma_1, \Gamma_2' \longrightarrow \Delta_1, \Delta_2; \Theta)$ By ind. hyp. on A, d, and e_1 from $(\mathcal{D}, B_1)$ and $\mathcal{E}_1$
$\mathcal{F} :: (\Psi; \Gamma_1, \Gamma_2', !B_1 \longrightarrow \Delta_1, \Delta_2; \Theta)$ By $!L$ from $\mathcal{E}_1'$

```
adr_!l : ad A D ([n] !l (E1 n) N) (!l E1' N)
          <- ({n1!:neg! B1} ad A D ([n] E1 n n1!) (E1' n1!)).
```

Commutative Cases of Cut!. Here we have a Cut! where the cut formula A is not the principal formula of $\mathcal{E}$. Note that the only way A could be the principal formula would be a structural rule $!D$, since otherwise non-linear hypothesis participate in inferences only as side formulas. Thus we have precisely one case for each possible inference rule R in $\mathcal{E}$, including axioms. In each case we simply commute the cut into the premises of $\mathcal{E}$ and then re-apply R to the results of appeals to the induction hyphotheses. Note that we do not distinguish cases based on the last inference in $\mathcal{D}$. Eventually, the Cut! disappears at the leaves of the derivation or is involved in a structural rule, in which case a structural case from above applies. We show only two of the mechanical commutative cases here.

Case: $\mathcal{D} :: (\Psi; \cdot \longrightarrow A; \Theta)$ is arbitrary and

$$\mathcal{E} = \frac{}{(\Psi, A); B \longrightarrow B; \Theta} I.$$

Then

$$\mathcal{F} = \frac{}{\Psi; B \longrightarrow B; \Theta} I$$

is a derivation with the desired end sequent.

```
ad!r_axiom : ad! A D! ([n!] axiom N P) (axiom N P).
```

Case: $\mathcal{D} :: (\Psi; \cdot \longrightarrow A; \Theta)$ is arbitrary and

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_1\\ (\Psi, A); \Gamma_2' \longrightarrow B_1, \Delta_2'; \Theta\end{array} \qquad \begin{array}{c}\mathcal{E}_2\\ (\Psi, A), \Gamma_2'' \longrightarrow B_2, \Delta_2''; \Theta\end{array}}{(\Psi, A), \Gamma_2', \Gamma_2'' \longrightarrow B_1 \otimes B_2, \Delta_2', \Delta_2''; \Theta} \otimes R.$$

Writing d, e_1 and e_2 for the proof terms of $\mathcal{D}$, $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, we obtain

$\mathcal{E}_1' :: (\Psi; \Gamma_2' \longrightarrow B_1, \Delta_2'; \Theta)$	By ind. hyp. on A, d, and e_1 from $\mathcal{D}$ and $\mathcal{E}_1$
$\mathcal{E}_2' :: (\Psi; \Gamma_2'' \longrightarrow B_2, \Delta_2''; \Theta)$	By ind. hyp. on A, d, and e_2 from $\mathcal{D}$ and $\mathcal{E}_2$
$\mathcal{F} :: (\Psi; \Gamma_2', \Gamma_2'' \longrightarrow B_1 \otimes B_2, \Delta_2', \Delta_2''; \Theta)$	By $\otimes R$ from $\mathcal{E}_1'$ and $\mathcal{E}_2'$.

```
ad!r_timesr : ad! A D! ([n!] timesr (E1 n!) (E2 n!) P)
               (timesr E1' E2' P)
               <- ({p1:pos B1} ad! A D! ([n!] E1 n! p1) (E1' p1))
               <- ({p2:pos B2} ad! A D! ([n!] E2 n! p2) (E2' p2)).
```

Commutative Cases of Cut?. Here we have a Cut? where the cut formula A is not the principal formula of $\mathcal{D}$. These cases are dual to the preceding commutative cases Cut!.

□

The faithful and concise implementation of this proof in the spirit of similar representations for intuitionistic and classical logic ([Pfe94]) would require a linear logical framework, which is the subject of current research. By meta-level reasoning (*i.e.*, the informal proof and properties of LF) we know that our implementation always maps linear derivations to linear derivations.

Corollary 9 (Admissibility of Cut for CLL) *The rule of cut is admissible in* CLL*.*

Proof: We translate the cut free derivations of the premises of the cut in CLL to LV (using Theorem 2) and apply Theorem 8. We then translate the resulting cut-free derivation to a cut-free derivation in CLL using Theorem 2 in the other direction. □

The translations between CLL and LV preserve enough structure that we could map our algorithm for admissibility of cut on LV to an algorithm for admissibility of cut on CLL. We do not pursue this here.

7 Cut Elimination

The theorem of cut elimination states that a derivation possibly using many cuts may be transformed into one without cuts. This follows by a simple structural induction from the admissibility of cut. We introduce a new system, LV^+ with primitive rules of cut and write $\Psi; \Gamma \longrightarrow_+ \Delta; \Theta$ for the system without proof terms and $\Psi; \Gamma \xrightarrow{d}_+ \Delta; \Theta$ for the system with proof terms. The latter contains the rules

$$\dfrac{\Psi; \Gamma_1 \xrightarrow{d}_+ p{:}A, \Delta_1; \Theta \qquad \Psi; \Gamma_2, n{:}A \xrightarrow{e}_+ \Delta_2; \Theta}{\Psi; \Gamma_1, \Gamma_2 \xrightarrow{\mathrm{cut}\,(\lambda p{:}A.\ d)\,(\lambda n{:}A.\ e)}_+ \Delta_1, \Delta_2; \Theta}\mathrm{Cut}$$

$$\dfrac{\Psi; \cdot \xrightarrow{d}_+ p{:}A; \Theta \qquad (\Psi, n^\omega{:}A); \Gamma \xrightarrow{e}_+ \Delta; \Theta}{\Psi; \Gamma \xrightarrow{\mathrm{cut!}\cdot(\lambda p{:}A.\ d)\,(\lambda^\omega n^\omega{:}A.\ e)}_+ \Delta; \Theta}\mathrm{Cut!}$$

$$\dfrac{\Psi; \Gamma \xrightarrow{d}_+ \Delta; (p^\omega{:}A, \Theta) \qquad \Psi; n{:}A \xrightarrow{e}_+ \cdot; \Theta}{\Psi; \Gamma \xrightarrow{\mathrm{cut?}\,(\lambda^\omega p^\omega{:}A.\ D)\cdot(\lambda n{:}A.\ e)}_+ \Delta; \Theta}\mathrm{Cut?}$$

Theorem 10 (Cut Elimination for LV^+) *If* $\Psi; \Gamma \xrightarrow{\hat{d}}_+ \Delta; \Theta$ *then* $\Psi; \Gamma \xrightarrow{d} \Delta; \Theta$ *for some proof term* d*.*

Proof: By structural induction on $\hat{d}$. For all rules R except the cut rules we appeal to induction hypotheses on the derivation(s) of the premise(s) and then apply R to the resulting cut-free derivation(s). In the case of a cut we first eliminate all cuts from the derivations of the premises (using the induction hypothesis) and then apply the admissibility of cut for the resulting cut-free derivations (Theorem 8) to obtain a cut-free derivation of the conclusion. □

Since the Elf language implementation currently supports neither subtyping nor modules, we implement LV^+ declaring a new type family `@` for derivations with cut and copying all the inference rules, appending `^` to their names to avoid name conflicts. The implementation of the cut elimination algorithm induced by our proof above is then straightforward. We only show the declarations, one congruence case, and the cut cases. The full implementation appears in Appendix A.7. Once again, the implementation does not check linearity constraints.

```
elim : @ -> # -> type.

ce_timesr : elim (timesr^ D1^ D2^ P) (timesr D1 D2 P)
             <- ({p1} elim (D1^ p1) (D1 p1))
             <- ({p2} elim (D2^ p2) (D2 p2)).

ce_cut    : elim (cut^ D1^ D2^) F
             <- ({p} elim (D1^ p) (D1 p))
             <- ({n} elim (D2^ n) (D2 n))
             <- ad A D1 D2 F.

ce_cut!   : elim (cut!^ D1!^ D2^) F
             <- ({p} elim (D1!^ p) (D1! p))
             <- ({n!} elim (D2^ n!) (D2 n!))
             <- ad! A D1! D2 F.

ce_cut?   : elim (cut?^ D1^ D2!^) F
             <- ({p?} elim (D1^ p?) (D1 p?))
             <- ({n} elim (D2!^ n) (D2! n))
             <- ad? A D1 D2! F.
```

We can obtain cut elimination for CLL in three ways: (1) we could prove it by structural induction from Corollary 9, (2) we could extend the translations from Theorem 2 to include cut and then use cut elimination for LV, or (3) we could obtain a direct algorithm by composing the translations with the cut elimination steps of LV. In any case, we have:

Corollary 11 (Cut Elimination for CLL**)** *The linear sequent calculus* CLL *satisfies cut elimination.*

8 Representation in a Linear Meta-Language

In this somewhat speculative section we show how to exploit a meta-language based on linear natural deduction in order to represent the linear sequent calculus more concisely. A full realization of this idea requires a dependently typed linear logical framework [MPP92] which is the subject of current research in joint work with I. Cervesato. In the absence of such a framework we limit ourselves to the propositional fragment of the linear sequent calculus LV, since its proof terms may be mapped rather directly onto the Lolli fragment of a linear natural deduction calculus [HM94]. We believe that the ideas can be extended naturally to a full calculus, including the proof of cut elimination. Throughout this section, we restrict ourselves to the propositional fragment LVP of LV.

The intuitionistic meta-language we need contains linear and intuitionistic implication, additive conjunction, and the additive unit. We endow its natural deduction formulation with proof terms.

$$\begin{array}{rrcll}
\text{Linear Types} & A & ::= & P \mid A_1 \multimap A_2 \mid A_1 \& A_2 \mid \top \mid A_1 \to A_2 & \\
\text{Linear Terms} & M & ::= & c \mid x & \\
 & & & \mid \lambda x{:}A.\ M \mid M_1\, M_2 & \text{for } A_1 \multimap A_2 \\
 & & & \mid \langle M_1, M_2 \rangle \mid \pi_1 M \mid \pi_2 M & \text{for } A_1 \& A_2 \\
 & & & \mid \langle\, \rangle & \text{for } \top \\
 & & & \mid \overset{\omega}{\lambda} x{:}A.\ M \mid M_1 \cdot M_2 & \text{for } A_1 \to A_2 \\
\text{Linear Contexts} & G & ::= & \cdot \mid G, x{:}A &
\end{array}$$

The overall context is divided into non-linear and linear zones. We use E for the non-linear context and L for the linear context, fixing a signature Σ for constants. We define here only the judgment for canonical forms (which correspond to so-called uniform derivations in [HM94]), because this is required for adequacy. We obtain the typing judgment for this linear λ-calculus by replacing both $\Uparrow$ and $\downarrow$ by : and omitting the (now redundant) rule which coerces from $\downarrow$ to $\Uparrow$. It follows from results in [Hod94] that every term can be transformed into canonical form. Note that the canonical forms correspond to long $\beta\eta$-normal forms.

$$\begin{array}{ll}
E; L \overset{L}{\vdash} M \Uparrow A & M \text{ is canonical of type } A \text{ in context } E; L \\
E; L \overset{L}{\vdash} M \downarrow A & M \text{ is atomic of type } A \text{ in context } E; L
\end{array}$$

It is defined by the following rules.

$$\frac{E; (L, x{:}A) \overset{L}{\vdash} M \Uparrow B}{E; L \overset{L}{\vdash} \lambda x{:}A.\ M \Uparrow A \multimap B} \qquad \frac{(E, x{:}A); L \overset{L}{\vdash} M \Uparrow B}{E; L \overset{L}{\vdash} \overset{\omega}{\lambda} x{:}A.\ M \Uparrow A \to B}$$

$$\frac{E; L \overset{L}{\vdash} M \Uparrow A \qquad E; L \overset{L}{\vdash} N \Uparrow B}{E; L \overset{L}{\vdash} \langle M, N \rangle \Uparrow A \& B} \qquad \frac{}{E; L \overset{L}{\vdash} \langle\, \rangle \Uparrow \top}$$

$$\frac{E; L \overset{L}{\vdash} M \downarrow P}{E; L \overset{L}{\vdash} M \Uparrow P} \qquad \frac{c{:}A \text{ in } \Sigma}{E; \cdot \overset{L}{\vdash} c \downarrow A}$$

$$\frac{}{E; x{:}A \overset{L}{\vdash} x \downarrow A} \qquad \frac{x{:}A \text{ in } E}{E; \cdot \overset{L}{\vdash} x \downarrow A}$$

$$\frac{E; L_1 \overset{L}{\vdash} M \downarrow A \multimap B \qquad E; L_2 \overset{L}{\vdash} N \Uparrow A}{E; L_1, L_2 \overset{L}{\vdash} M\, N \downarrow B} \qquad \frac{E; L \overset{L}{\vdash} M \downarrow A \to B \qquad E; \cdot \overset{L}{\vdash} N \Uparrow A}{E; L \overset{L}{\vdash} M \cdot N \downarrow B}$$

$$\frac{E; L \overset{L}{\vdash} M \downarrow A \& B}{E; L \overset{L}{\vdash} \pi_1 M \downarrow A} \qquad \frac{E; L \overset{L}{\vdash} M \downarrow A \& B}{E; L \overset{L}{\vdash} \pi_2 M \downarrow B}$$

Below we give the declarations of the constants that could be used to construct the linear λ-terms to obtain an adequate representation of sequent derivations in LV. Alternatively, one can think of this as a Lolli program, where each clause has been labelled.

```
# : type.         % Token for Derivations
neg!: o -> type.  % Modal Hypotheses (far left)
neg : o -> type.  % Hypotheses (left)
pos : o -> type.  % Conclusions (right)
pos?: o -> type.  % Modal Conclusions (far right)

axiom : (neg A -o pos A -o #).

timesr : (pos A -o #)                    timesl : (neg A -o neg B -o #)
          -o (pos B -o #)                          -o (neg (A times B) -o #).
          -o (pos (A times B) -o #).

oner : (pos one -o #).                   onel : # -o (neg one -o #).

                                         andl1 : (neg A -o #)
andr : ((pos A -o #) & (pos B -o #))              -o (neg (A and B) -o #).
        -o (pos (A and B) -o #).
                                         andl2 : (neg B -o #)
                                                  -o (neg (A and B) -o #).

topr : T -o (pos (top) -o #).            % no topl

perpr : (neg A -o #)                     perpl : (pos A -o #)
         -o (pos (perp A) -o #).                  -o (neg (perp A) -o #).

!r : (pos A -o #)                        !l : (neg! A -> #)
      -> (pos (! A) -o #).                     -o (neg (! A) -o #).

?r : (pos? A -> #)                       ?l : (neg A -o #)
      -o (pos (? A) -o #).                     -> (neg (? A) -o #).

!d : (neg A -o #)                        ?d : (pos A -o #)
      -o (neg! A -> #).                        -o (pos? A -> #).
```

If the cut rules were primitive, they would be represented by

```
cut : (pos A -o #)
       -o (neg A -o #)
       -o #.

cut! : (pos A -o #)
        -> (neg! A -> #)
        -o #.

cut? : (neg? A -> #)
        -o (pos A -o #)
        -> #.
```

Since the framework can now express linearity constraints directly, we no longer require the judgments `linp`, `linn`, and `lin` from before. However, the ambiguities of proof terms must still be taken into account when formulating the adequacy theorem. Proof terms from LVP are represented directly by linear terms in the meta-language. Sequent derivations are mapped to derivations showing that these terms in the meta-language are canonical (and thereby, that they satisfy linearity constraints). We write $\ulcorner\mathcal{D}\urcorner$ for this meta-derivation. We do not write out its definition, since it follows obviously from the representation of proof terms. The main point of this mapping is that linear hypotheses and conclusions of the sequent derivation $\mathcal{D}$ are distributed to the premises in an analogous manner in $\ulcorner\mathcal{D}\urcorner$.

Lemma 12 (Soundness of Linear Representation) *Let $\mathcal{D} :: (\Psi; \Gamma \xrightarrow{d} \Delta; \Theta)$ be a sequent derivation in* LVP. *Then*

$$\ulcorner\mathcal{D}\urcorner :: ((\ulcorner\Psi\urcorner, \ulcorner\Theta\urcorner); (\ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner) \overset{L}{\vdash} \ulcorner d\urcorner \Uparrow \#).$$

Proof: By induction on the structure of $\mathcal{D}$. □

Lemma 13 (Completeness of Linear Representation) *Let E be a non-linear context of the form $\ulcorner\Psi\urcorner, \ulcorner\Theta\urcorner$ and L be of the form $\ulcorner\Gamma\urcorner, \ulcorner\Delta\urcorner$.*

1. *If $E; L \overset{L}{\vdash} M \Uparrow \#$ then there exists a unique proof term d such that $\ulcorner d\urcorner = M$.*
2. *For every derivation $\mathcal{L} :: (E; L \overset{L}{\vdash} M \Uparrow \#)$ there exists a unique* LVP *sequent derivation $\mathcal{D} :: (\Psi; \Gamma \xrightarrow{d} \Delta; \Theta)$ such that $\ulcorner\mathcal{D}\urcorner = \mathcal{L}$.*

Proof: By induction on the structure of $\mathcal{L}$. □

Theorem 14 (Adequacy of Linear Representation) *The representation function $\ulcorner\cdot\urcorner$ on sequent calculus proof terms is a bijection between valid proof terms d and canonical terms M of type $\#$ in the appropriate contexts. Moreover, $\ulcorner\cdot\urcorner$ on sequent derivations is a bijection between sequent derivations $\mathcal{D}$ with proof term d and derivations $\mathcal{L}$ showing that $\ulcorner d\urcorner$ is canonical.*

9 Conclusion

We presented LV, a sequent formulation of classical linear logic with proof terms and gave a structural proof of cut elimination for it. We also presented an adequate encoding of linear sequent derivations in LF and an implementation of a cut elimination algorithm for LV in Elf.

The implementation of cut elimination is operationally adequate, but it is not a complete implementation of the cut elimination proof, since it disregards all linearity constraints. We believe it would be possible, albeit *very* tedious (without any automation) to implement a proof of the property that our algorithms maps linear derivations to linear derivations.

This naturally leads to the question if the proof of cut elimination could be represented faithfully and directly in a *linear logical framework* along the lines of [MPP92]. We conjecture that this is possible and plan to investigate it in future work. Preliminary evidence in this direction is a more direct encoding of sequent derivations themselves presented in Section 8.

Another interesting direction for further research would be to reformulate the algorithm into a set of higher-order rewrite rules on linear λ-terms. The resulting system appears to be canonical modulo permutations of adjacent inferences as analyzed by Galmiche & Perrier [GP]. We have checked this mechanically only for the intuitionistic and classical (non-linear) version of our cut elimination procedure as presented in [Pfe94].

Acknowledgments

I would like to thank Iliano Cervesato for discussions regarding the topic of this paper and proof-reading earlier drafts.

A Complete Implementation

A.1 Formulas

```
%file formulas.elf

i : type.  % individuals
o : type.  % formulas
%name i S
%name o A B C

% Multiplicative connectives
times  : o -> o -> o.  %infix right 11 times
one    : o.

% Additive connectives
and    : o -> o -> o.  %infix right 11 and
top    : o.

% Involution
perp   : o -> o.

% Quantifier
forall : (i -> o) -> o.

% Exponentials
!      : o -> o.
?      : o -> o.
```

A.2 Sequent Calculus LV

```
%file lv.elf

# : type.         % Token (for derivations)
neg!: o -> type.  % Exponential Hypotheses (far left zone)
neg : o -> type.  % Hypotheses (left zone)
pos : o -> type.  % Conclusions (right zone)
pos?: o -> type.  % Exponential Conclusions (far right zone)
%name # D
%name neg! N!
%name neg N
%name pos P
%name pos? N?

%% Axioms

axiom : (neg A -> pos A -> #).

%% Multiplicative Connectives

timesr : (pos A -> #)
          -> (pos B -> #)
```

```
          -> (pos (A times B) -> #).

timesl : (neg A -> neg B -> #)
          -> (neg (A times B) -> #).

oner : (pos one -> #).

onel : #
        -> (neg one -> #).

%% Additive Connectives

andr : (pos A -> #) -> (pos B -> #)
        -> (pos (A and B) -> #).

andl1 : (neg A -> #)
         -> (neg (A and B) -> #).

andl2 : (neg B -> #)
         -> (neg (A and B) -> #).

topr : (pos (top) -> #).

% no topl

%% Involution

perpr : (neg A -> #)
         -> (pos (perp A) -> #).

perpl : (pos A -> #)
         -> (neg (perp A) -> #).

%% Quantifier

forallr : ({a:i} pos (A a) -> #)
           -> (pos (forall A) -> #).

foralll : {T:i} (neg (A T) -> #)
           -> (neg (forall A) -> #).

%% Exponentials

!r : (pos A -> #)
      -> (pos (! A) -> #).

!l : (neg! A -> #)
      -> (neg (! A) -> #).

!d : (neg A -> #)
      -> (neg! A -> #).
```

```
?r : (pos? A -> #)
      -> (pos (? A) -> #).

?l : (neg A -> #)
      -> (neg (? A) -> #).

?d : (pos A -> #)
      -> (pos? A -> #).
```

A.3 Linearity Constraints

```
%file linear.elf

%% Linear functions in positive formula

linp : (pos A -> #) -> type.

%mode -linp +D
%lex D

linp_axiom    : linp ([p] axiom N p).

linp_timesr_0 : linp ([p] timesr D1 D2 p).

linp_timesr_1 : linp ([p] timesr (D1 p) D2 P)
                <- ({p1} linp ([p] D1 p p1)).

linp_timesr_2 : linp ([p] timesr D1 (D2 p) P)
                <- ({p2} linp ([p] D2 p p2)).

linp_timesl   : linp ([p] timesl (D1 p) N)
                <- ({n1} {n2} linp ([p] D1 p n1 n2)).

linp_oner_0   : linp ([p] oner p).

linp_onel     : linp ([p] onel (D1 p) N)
                <- linp ([p] D1 p).

linp_andr_0   : linp ([p] andr D1 D2 p).

linp_andr     : linp ([p] andr (D1 p) (D2 p) P)
                 <- ({p1} linp ([p] D1 p p1))
                 <- ({p2} linp ([p] D2 p p2)).

linp_andl1    : linp ([p] andl1 (D1 p) N)
                 <- ({n1} linp ([p] D1 p n1)).

linp_andl2    : linp ([p] andl2 (D2 p) N)
                 <- ({n2} linp ([p] D2 p n2)).
```

```
linp_topr_0   : linp ([p] topr p).

linp_topr     : linp ([p] topr P).

linp_perpr_0  : linp ([p] perpr D1 p).

linp_perpr    : linp ([p] perpr (D1 p) P)
                 <- ({n1} linp ([p] D1 p n1)).

linp_perpl    : linp ([p] perpl (D1 p) N)
                 <- ({p1} linp ([p] D1 p p1)).

linp_forallr_0 : linp ([p] forallr D1 p).

linp_forallr  : linp ([p] forallr (D1 p) P)
                 <- ({a} {p1} linp ([p] D1 p a p1)).

linp_foralll  : linp ([p] foralll T (D1 p) N)
                 <- ({n1} linp ([p] D1 p n1)).

linp_!r_0     : linp ([p] !r D1 p).

% no linp_!r: p may not occur in D1.

linp_!l       : linp ([p] !l (D1! p) N)
                 <- ({n1!} linp ([p] D1! p n1!)).

linp_!d       : linp ([p] !d (D1 p) N!)
                 <- ({n1} linp ([p] D1 p n1)).

linp_?r_0     : linp ([p] ?r D1 p).

linp_?r       : linp ([p] ?r (D1? p) P)
                 <- ({p1?} linp ([p] D1? p p1?)).

% no linp_?l_0 : p is positive
% no linp_?l: p may not occur in D1

linp_?d       : linp ([p] ?d (D1 p) P?)
                 <- ({p1} linp ([p] D1 p p1)).

%% Linear function in negative formula

linn : (neg A -> #) -> type.

%mode -linn +D
%lex D

linn_axiom    : linn ([n] axiom n P).

linn_timesr_1 : linn ([n] timesr (D1 n) D2 P)
```

```
                <- ({p1} linn ([n] D1 n p1)).

linn_timesr_2 : linn ([n] timesr D1 (D2 n) P)
                <- ({p2} linn ([n] D2 n p2)).

linn_timesl_0 : linn ([n] timesl D1 n).

linn_timesl   : linn ([n] timesl (D1 n) N)
                <- ({n1} {n2} linn ([n] D1 n n1 n2)).

% no linn_oner

linn_onel_0   : linn ([n] onel D1 n).

linn_onel     : linn ([n] onel (D1 n) N)
                <- linn ([n] D1 n).

linn_andr     : linn ([n] andr (D1 n) (D2 n) P)
                 <- ({p1} linn ([n] D1 n p1))
                 <- ({p2} linn ([n] D2 n p2)).

linn_andl1_0  : linn ([n] andl1 D1 n).

linn_andl1    : linn ([n] andl1 (D1 n) N)
                 <- ({n1} linn ([n] D1 n n1)).

linn_andl2_0  : linn ([n] andl2 D2 n).

linn_andl2    : linn ([n] andl2 (D2 n) N)
                 <- ({n2} linn ([n] D2 n n2)).

linn_topr     : linn ([n] topr P).

linn_perpr    : linn ([n] perpr (D1 n) P)
                 <- ({n1} linn ([n] D1 n n1)).

linn_perpl_0  : linn ([n] perpl D1 n).

linn_perpl    : linn ([n] perpl (D1 n) N)
                 <- ({p1} linn ([n] D1 n p1)).

linn_forallr  : linn ([n] forallr (D1 n) P)
                 <- ({a} {p1} linn ([n] D1 n a p1)).

linn_foralll_0 : linn ([n] foralll T D1 n).

linn_foralll  : linn ([n] foralll T (D1 n) N)
                 <- ({n1} linn ([n] D1 n n1)).

% no linn_!r_0 : n is negative
% no linn_!r: n may not occur in D1.
```

```
linn_!l_0     : linn ([n] !l D1! n).

linn_!l       : linn ([n] !l (D1! n) N)
                 <- ({n1!} linn ([n] D1! n n1!)).

linn_!d       : linn ([n] !d (D1 n) N!)
                 <- ({n1} linn ([n] D1 n n1)).

linn_?r       : linn ([n] ?r (D1? n) P)
                 <- ({p1?} linn ([n] D1? n p1?)).

linn_?l_0     : linn ([n] ?l D1 n).

% no linn_?l: n may not occur in D1

linn_?d       : linn ([n] ?d (D1 n) P?)
                 <- ({p1} linn ([n] D1 n p1)).

%% Linear derivations: all subderivations must be
%% linear as required.

lin  : # -> type.

%mode -lin +D
%lex D

lin_axiom  : lin (axiom N P).

lin_timesr : lin (timesr D1 D2 P)
              <- linp D1
              <- linp D2
              <- ({p1} lin (D1 p1))
              <- ({p2} lin (D2 p2)).

lin_timesl : lin (timesl D1 N)
              <- ({n2} linn ([n1] D1 n1 n2))
              <- ({n1} linn ([n2] D1 n1 n2))
              <- ({n1} {n2} lin (D1 n1 n2)).

lin_oner   : lin (oner P).

lin_onel   : lin (onel D N)
              <- lin D.

lin_andr   : lin (andr D1 D2 P)
              <- linp D1
              <- linp D2
              <- ({p1} lin (D1 p1))
              <- ({p2} lin (D2 p2)).
```

```
lin_andl1  : lin (andl1 D1 N)
              <- linn D1
              <- ({n1} lin (D1 n1)).

lin_andl2  : lin (andl2 D2 N)
              <- linn D2
              <- ({n2} lin (D2 n2)).

lin_topr   : lin (topr P).

% no topl

lin_perpr  : lin (perpr D1 P)
              <- linn D1
              <- ({n1} lin (D1 n1)).

lin_perpl  : lin (perpl D1 N)
              <- linp D1
              <- ({p1} lin (D1 p1)).

lin_forallr : lin (forallr D1 P)
               <- ({a:i} linp (D1 a))
               <- ({a} {p1} lin (D1 a p1)).

lin_foralll : lin (foralll T D1 N)
               <- linn D1
               <- ({n1} lin (D1 n1)).

lin_!r     : lin (!r D1 P)
              <- linp D1
              <- ({p1} lin (D1 p1)).

lin_!l     : lin (!l D1! N)
              <- ({n1!} lin (D1! n1!)).

lin_!d     : lin (!d D1 N!)
              <- linn D1
              <- ({n1} lin (D1 n1)).

lin_?r     : lin (?r D1? P)
              <- ({p1?} lin (D1? p1?)).

lin_?l     : lin (?l D1 N)
              <- linn D1
              <- ({n1} lin (D1 n1)).

lin_?d     : lin (?d D1 P?)
              <- linp D1
              <- ({p1} lin (D1 p1)).

%% Linear derivation of  B --> C  (common case).
```

```
lin2 : (neg B -> pos C -> #) -> type.

%mode -lin2 +D
%lex D

lin2_all : lin2 D
             <- ({p} linn ([n] D n p))  % linear in n
             <- ({n} linp ([p] D n p))  % linear in p
             <- ({n} {p} lin (D n p)).  % subderivations are all linear
```

A.4 Admissibility of Cut

```
%file admit.elf

ad  : {A:o} (pos A -> #) -> (neg A -> #) -> # -> type.
ad! : {A:o} (pos A -> #) -> (neg! A -> #) -> # -> type.
ad? : {A:o} (pos? A -> #) -> (neg A -> #) -> # -> type.

%mode -ad! +A1 +D1! +E1  -F1
%mode -ad? +A2 +D2  +E2! -F2
%mode -ad  +A3 +D3  +E3  -F3

%lex {A1 A2 A3} {} {D1! D2 D3} {E1 E2! E3}

%% 1. Axiom Cases

ad_axiom_l : ad A ([p] axiom N p) E (E N).

ad_axiom_r : ad A D ([n] axiom n P) (D P).

%% 2. Essential Cases

% Essential cases, multiplicatives

ad_times : ad (A1 times A2) ([p] timesr D1 D2 p) ([n] timesl E1 n) F
            <- ({n2:neg A2} ad A1 D1 ([n1] E1 n1 n2) (E1' n2))
            <- ad A2 D2 E1' F.

ad_one   : ad (one) ([p] oner p) ([n] onel E1 n) E1.

% Essential cases, additives

ad_and1 : ad (A1 and A2) ([p] andr D1 D2 p) ([n] andl1 E1 n) F
           <- ad A1 D1 E1 F.

ad_and2 : ad (A1 and A2) ([p] andr D1 D2 p) ([n] andl2 E2 n) F
           <- ad A2 D2 E2 F.

% No essential case for additive unit
```

```
% Essentional case, involution

ad_perp : ad (perp A1) ([p] perpr D1 p) ([n] perpl E1 n) F
           <- ad A1 E1 D1 F.

% Essential case, quantifier

ad_forall : ad (forall A1) ([p] forallr D1 p) ([n] foralll T E1 n) F
             <- ad (A1 T) (D1 T) E1 F.

% Essential cases, exponentials

ad_! : ad (! A1) ([p] !r D1! p) ([n] !l E1 n) F
        <- ad! A1 D1! E1 F.

ad_? : ad (? A1) ([p] ?r D1 p) ([n] ?l E1! n) F
        <- ad? A1 D1 E1! F.

%% 3. Structural Cases

ad!_d  : ad! A D! ([n!] !d (E1 n!) n!) F
          <- ({n1:neg A} ad! A D! ([n!] E1 n! n1) (E1' n1))
          <- ad A D! E1' F.

ad?_r  : ad? A ([p?] ?d (D1 p?) p?) E! F
          <- ({p1:pos A} ad? A ([p?] D1 p? p1) E! (D1' p1))
          <- ad A D1' E! F.

%% 4. Commutative Cases

% 4.1 Right commutative cases of Cut

% No commutative cases crossing axioms

% Crossing multiplicatives

adr_timesr1 : ad A D ([n] timesr (E1 n) E2 P) (timesr E1' E2 P)
              <- ({p1:pos B1} ad A D ([n] E1 n p1) (E1' p1)).

adr_timesr2 : ad A D ([n] timesr E1 (E2 n) P) (timesr E1 E2' P)
              <- ({p2:pos B2} ad A D ([n] E2 n p2) (E2' p2)).

adr_timesl  : ad A D ([n] timesl (E1 n) N) (timesl E1' N)
              <- ({n1:neg B1} {n2:neg B2}
                    ad A D ([n] E1 n n1 n2) (E1' n1 n2)).

% No case for oner

adr_onel    : ad A D ([n] onel (E1 n) N) (onel E1' N)
              <- ad A D E1 E1'.
```

```
% Crossing additives

adr_andr : ad A D ([n] andr (E1 n) (E2 n) P) (andr E1' E2' P)
           <- ({p1:pos B1} ad A D ([n] E1 n p1) (E1' p1))
           <- ({p2:pos B2} ad A D ([n] E2 n p2) (E2' p2)).

adr_andl1 : ad A D ([n] andl1 (E1 n) N) (andl1 E1' N)
             <- ({n1:neg B1} ad A D ([n] E1 n n1) (E1' n1)).

adr_andl2 : ad A D ([n] andl2 (E2 n) N) (andl2 E2' N)
             <- ({n2:neg B2} ad A D ([n] E2 n n2) (E2' n2)).

adr_topr  : ad A D ([n] topr P) (topr P).

% no topl rule

% Crossing involution

adr_perpr : ad A D ([n] perpr (E1 n) P) (perpr E1' P)
             <- ({n1:neg B1} ad A D ([n] E1 n n1) (E1' n1)).

adr_perpl : ad A D ([n] perpl (E1 n) N) (perpl E1' N)
             <- ({p1:pos B1} ad A D ([n] E1 n p1) (E1' p1)).

% Crossing quantifier

adr_forallr : ad A D ([n] forallr (E1 n) P) (forallr E1' P)
               <- ({a:i} {p1:pos (B1 a)} ad A D ([n] E1 n a p1) (E1' a p1)).

adr_foralll : ad A D ([n] foralll T (E1 n) N) (foralll T E1' N)
               <- ({n1} ad A D ([n] E1 n n1) (E1' n1)).

% Crossing exponentials

% no adr_!r since there are no linear side formulas

% Meta-level weakening is required in next case
adr_!l : ad A D ([n] !l (E1 n) N) (!l E1' N)
          <- ({n1!:neg! B1} ad A D ([n] E1 n n1!) (E1' n1!)).

adr_!d : ad A D ([n] !d (E1 n) N!) (!d E1' N!)
          <- ({n1:neg B} ad A D ([n] E1 n n1) (E1' n1)).

% Meta-level weakening is required in next case
adr_?r : ad A D ([n] ?r (E1 n) P) (?r E1' P)
          <- ({p1?:pos? B1} ad A D ([n] E1 n p1?) (E1' p1?)).

% no adr_?l since there are no linear side formulas

adr_?d : ad A D ([n] ?d (E1 n) P?) (?d E1' P?)
          <- ({p1:pos B} ad A D ([n] E1 n p1) (E1' p1)).
```

```
% 4.2 Left commutative cases of Cut

% No commutative cases crossing axioms

% Crossing multiplicatives

adl_timesr1 : ad A ([p] timesr (D1 p) D2 P) E (timesr D1' D2 P)
               <- ({p1:pos B1} ad A ([p] D1 p p1) E (D1' p1)).

adl_timesr2 : ad A ([p] timesr D1 (D2 p) P) E (timesr D1 D2' P)
               <- ({p2:pos B2} ad A ([p] D2 p p2) E (D2' p2)).

adl_timesl  : ad A ([p] timesl (D1 p) N) E (timesl D1' N)
               <- ({n1:neg B1} {n2:neg B2}
                     ad A ([p] D1 p n1 n2) E (D1' n1 n2)).

% No case for oner

adl_onel    : ad A ([p] onel (D1 p) N) E (onel D1' N)
               <- ad A D1 E D1'.

% Crossing additives

adl_andr  : ad A ([p] andr (D1 p) (D2 p) P) E (andr D1' D2' P)
             <- ({p1:pos B1} ad A ([p] D1 p p1) E (D1' p1))
             <- ({p2:pos B2} ad A ([p] D2 p p2) E (D2' p2)).

adl_andl1 : ad A ([p] andl1 (D1 p) N) E (andl1 D1' N)
             <- ({n1:neg B1} ad A ([p] D1 p n1) E (D1' n1)).

adl_andl2 : ad A ([p] andl2 (D2 p) N) E (andl2 D2' N)
             <- ({n2:neg B1} ad A ([p] D2 p n2) E (D2' n2)).

adl_topr  : ad A ([p] topr P) E (topr P).

% no topl rule

% Crossing involution

adl_perpr : ad A ([p] perpr (D1 p) P) E (perpr D1' P)
             <- ({n1:neg B1} ad A ([p] D1 p n1) E (D1' n1)).

adl_perpl : ad A ([p] perpl (D1 p) N) E (perpl D1' N)
             <- ({p1:pos B1} ad A ([p] D1 p p1) E (D1' p1)).

% Crossing quantifier

adl_forallr : ad A ([p] forallr (D1 p) P) E (forallr D1' P)
               <- ({a:i} {p1:pos (B1 a)} ad A ([p] D1 p a p1) E (D1' a p1)).
```

```
adl_foralll : ad A ([p] foralll T (D1 p) N) E (foralll T D1' N)
               <- ({n1} ad A ([p] D1 p n1) E (D1' n1)).

% Crossing exponentials

% no adl_!r since there are no linear side formulas

adl_!l : ad A ([p] !l (D1 p) N) E (!l D1' N)
          <- ({n1!:neg! B1} ad A ([p] D1 p n1!) E (D1' n1!)).

adl_!d : ad A ([p] !d (D1 p) N!) E (!d D1' N!)
          <- ({n1:neg B} ad A ([p] D1 p n1) E (D1' n1)).

adl_?r : ad A ([p] ?r (D1 p) P) E (?r D1' P)
          <- ({p1?:pos? B1} ad A ([p] D1 p p1?) E (D1' p1?)).

% no adl_?l since there are no linear side formulas

adl_?d : ad A ([p] ?d (D1 p) P?) E (?d D1' P?)
          <- ({p1:pos B} ad A ([p] D1 p p1) E (D1' p1)).

% 4.3 Right commutative cases of Cut!

% Crossing axioms

ad!r_axiom : ad! A D! ([n!] axiom N P) (axiom N P).

% Crossing multiplicatives

ad!r_timesr : ad! A D! ([n!] timesr (E1 n!) (E2 n!) P)
               (timesr E1' E2' P)
               <- ({p1:pos B1} ad! A D! ([n!] E1 n! p1) (E1' p1))
               <- ({p2:pos B2} ad! A D! ([n!] E2 n! p2) (E2' p2)).

ad!r_timesl : ad! A D! ([n!] timesl (E1 n!) N) (timesl E1' N)
               <- ({n1:neg B1} {n2:neg B2}
                     ad! A D! ([n!] E1 n! n1 n2) (E1' n1 n2)).

ad!r_oner   : ad! A D! ([n!] oner P) (oner P).

ad!r_onel   : ad! A D! ([n!] onel (E1 n!) N) (onel E1' N)
               <- ad! A D! E1 E1'.

% Crossing additives

ad!r_andr   : ad! A D! ([n!] andr (E1 n!) (E2 n!) P) (andr E1' E2' P)
               <- ({p1:pos B1} ad! A D! ([n!] E1 n! p1) (E1' p1))
               <- ({p2:pos B2} ad! A D! ([n!] E2 n! p2) (E2' p2)).

ad!r_andl1  : ad! A D! ([n!] andl1 (E1 n!) N) (andl1 E1' N)
               <- ({n1:neg B1} ad! A D! ([n!] E1 n! n1) (E1' n1)).
```

```
ad!r_andl2  : ad! A D! ([n!] andl2 (E2 n!) N) (andl1 E2' N)
               <- ({n2:neg B2} ad! A D! ([n!] E2 n! n2) (E2' n2)).

ad!r_topr   : ad! A D! ([n!] topr P) (topr P).

% no topl rule

% Crossing involutions

ad!r_perpr  : ad! A D! ([n!] perpr (E1 n!) P) (perpr E1' P)
               <- ({n1:neg B1} ad! A D! ([n!] E1 n! n1) (E1' n1)).

ad!r_perpl  : ad! A D! ([n!] perpl (E1 n!) N) (perpl E1' N)
               <- ({p1:pos B1} ad! A D! ([n!] E1 n! p1) (E1' p1)).

% Crossing quantifiers

ad!r_forallr : ad! A D! ([n!] forallr (E1 n!) P) (forallr E1' P)
                <- ({a:i} {p1:pos (B1 a)}
                      ad! A D! ([n!] E1 n! a p1) (E1' a p1)).

ad!r_foralll : ad! A D! ([n!] foralll T (E1 n!) N) (foralll T E1' N)
                <- ({n1} ad! A D! ([n!] E1 n! n1) (E1' n1)).

% Crossing exponentials

ad!r_!r : ad! A D! ([n!] !r (E1! n!) P) (!r E1!' P)
           <- ({p1:pos B1} ad! A D! ([n!] E1! n! p1) (E1!' p1)).

ad!r_!l : ad! A D! ([n!] !l (E1 n!) N) (!l E1' N)
           <- ({n1!:neg! B1} ad! A D! ([n!] E1 n! n1!) (E1' n1!)).

ad!r_!d : ad! A D! ([n!] !d (E1 n!) N!) (!d E1' N!)
           <- ({n1:neg B} ad! A D! ([n!] E1 n! n1) (E1' n1)).

ad!r_?r : ad! A D! ([n!] ?r (E1 n!) P) (?r E1' P)
           <- ({p1?:pos? B1} ad! A D! ([n!] E1 n! p1?) (E1' p1?)).

ad!r_?l : ad! A D! ([n!] ?l (E1! n!) N) (?l E1!' N)
           <- ({n1:neg B1} ad! A D! ([n!] E1! n! n1) (E1!' n1)).

ad!r_?d : ad! A D! ([n!] ?d (E1 n!) P?) (?d E1' P?)
           <- ({p1:pos B} ad! A D! ([n!] E1 n! p1) (E1' p1)).

% 4.4 No left commutative cases for Cut!
% Right commutative or structural
% cases will always be applicable for ad!

% 4.5 No right commutative cases for Cut?
% Left commutative cases or structural
```

```
% cases will always be applicable for ad?

% 4.6 Left commutative cases for Cut?

% Crossing axioms

ad?l_axiom : ad? A ([p?] axiom N P) E! (axiom N P).

% Crossing multiplicatives

ad?l_timesr : ad? A ([p?] timesr (D1 p?) (D2 p?) P) E! (timesr D1' D2' P)
               <- ({p1:pos B1} ad? A ([p?] D1 p? p1) E! (D1' p1))
               <- ({p2:pos B2} ad? A ([p?] D2 p? p2) E! (D2' p2)).

ad?l_timesl : ad? A ([p?] timesl (D1 p?) N) E! (timesl D1' N)
               <- ({n1:neg B1} {n2:neg B2}
                     ad? A ([p?] D1 p? n1 n2) E! (D1' n1 n2)).

ad?l_oner   : ad? A ([p?] oner P) E! (oner P).

ad?l_onel   : ad? A ([p?] onel (D1 p?) N) E! (onel D1' N)
               <- ad? A D1 E! D1'.

% Crossing additives

ad?l_andr  : ad? A ([p?] andr (D1 p?) (D2 p?) P) E! (andr D1' D2' P)
              <- ({p1:pos B1} ad? A ([p?] D1 p? p1) E! (D1' p1))
              <- ({p2:pos B2} ad? A ([p?] D2 p? p2) E! (D2' p2)).

ad?l_andl1 : ad? A ([p?] andl1 (D1 p?) N) E! (andl1 D1' N)
              <- ({n1:neg B1} ad? A ([p?] D1 p? n1) E! (D1' n1)).

ad?l_andl2 : ad? A ([p?] andl2 (D2 p?) N) E! (andl2 D2' N)
              <- ({n2:neg B2} ad? A ([p?] D2 p? n2) E! (D2' n2)).

ad?l_topr  : ad? A ([p?] topr P) E! (topr P).

% no topl rule

% Crossing involutions

ad?l_perpr : ad? A ([p?] perpr (D1 p?) P) E! (perpr D1' P)
              <- ({n1:neg B1} ad? A ([p?] D1 p? n1) E! (D1' n1)).

ad?l_perpl : ad? A ([p?] perpl (D1 p?) N) E! (perpl D1' N)
              <- ({p1:pos B1} ad? A ([p?] D1 p? p1) E! (D1' p1)).

% Crossing quantifiers

ad?l_forallr : ad? A ([p?] forallr (D1 p?) P) E! (forallr D1' P)
                <- ({a:i} {p1:pos (B1 a)}
```

```
                    ad? A ([p?] D1 p? a p1) E! (D1' a p1)).

ad?l_foralll : ad? A ([p?] foralll T (D1 p?) N) E! (foralll T D1' N)
                <- ({n1} ad? A ([p?] D1 p? n1) E! (D1' n1)).

% Crossing exponentials

ad?l_!r : ad? A ([p?] !r (D1! p?) P) E! (!r  D1!' P)
           <- ({p1:pos B1} ad? A ([p?] D1! p? p1) E! (D1!' p1)).

ad?l_!l : ad? A ([p?] !l (D1 p?) N) E! (!l D1' N)
           <- ({n1!:neg! B1} ad? A ([p?] D1 p? n1!) E! (D1' n1!)).

ad?l_!d : ad? A ([p?] !d (D1 p?) N!) E! (!d D1' N!)
           <- ({n1:neg B} ad? A ([p?] D1 p? n1) E! (D1' n1)).

ad?l_?r : ad? A ([p?] ?r (D1 p?) P) E! (?r D1' P)
           <- ({p1?:pos? B1} ad? A ([p?] D1 p? p1?) E! (D1' p1?)).

ad?l_?l : ad? A ([p?] ?l (D1! p?) N) E! (?l D1!' N)
           <- ({n1:neg B1} ad? A ([p?] D1! p? n1) E! (D1!' n1)).

ad?l_?d : ad? A ([p?] ?d (D1 p?) P?) E! (?d D1' P?)
           <- ({p1:pos B} ad? A ([p?] D1 p? p1) E! (D1' p1)).
```

A.5 Example Execution of Admissibility Algorithm

In this subsection we give derivations of $!(A \multimap B) \longrightarrow\ !A \multimap\ !B$ and $!A \multimap\ !B \longrightarrow\ ?(B^\perp) \multimap\ ?(A^\perp)$ and then apply admissibility of cut to obtain a derivation of $!(A \multimap B) \longrightarrow\ ?(B^\perp) \multimap\ ?(A^\perp)$. These derivations are complicated by the fact that $A \multimap B$ is defined as $(A \otimes B^\perp)^\perp$. We omit empty exponential hypotheses or conclusions and abbreviate $\Psi_0 = (A \otimes B^\perp)^\perp, A$.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$\Psi_0; A \longrightarrow A$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$\Psi_0; B \longrightarrow B$}
\RightLabel{$\neg R$}
\UnaryInfC{$\Psi_0; \cdot \longrightarrow B^\perp, B$}
\RightLabel{$\otimes R$}
\BinaryInfC{$\Psi_0; A \longrightarrow A \otimes B^\perp, B$}
\RightLabel{$\neg L$}
\UnaryInfC{$\Psi_0; (A \otimes B^\perp)^\perp, A \longrightarrow B$}
\RightLabel{$!D$}
\UnaryInfC{$\Psi_0; A \longrightarrow B$}
\RightLabel{$!D$}
\UnaryInfC{$\Psi_0; \cdot \longrightarrow B$}
\RightLabel{$!R$}
\UnaryInfC{$(A \otimes B^\perp)^\perp, A; \cdot \longrightarrow\ !B$}
\RightLabel{$!L$}
\UnaryInfC{$(A \otimes B^\perp)^\perp; !A \longrightarrow\ !B$}
\RightLabel{$!L$}
\UnaryInfC{$!((A \otimes B^\perp)^\perp), !A \longrightarrow\ !B$}
\RightLabel{$\neg L$}
\UnaryInfC{$!((A \otimes B^\perp)^\perp), !A, (!B)^\perp \longrightarrow \cdot$}
\RightLabel{$\otimes L$}
\UnaryInfC{$!((A \otimes B^\perp)^\perp), !A \otimes (!B)^\perp \longrightarrow \cdot$}
\RightLabel{$\neg R$}
\UnaryInfC{$!((A \otimes B^\perp)^\perp) \longrightarrow (!A \otimes (!B)^\perp)^\perp$}
\end{prooftree}
$$

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$A \longrightarrow A; A^\perp$}
\RightLabel{$\neg R$}
\UnaryInfC{$\cdot \longrightarrow A, A^\perp; A^\perp$}
\RightLabel{$?D$}
\UnaryInfC{$\cdot \longrightarrow A; A^\perp$}
\RightLabel{$!R$}
\UnaryInfC{$\cdot \longrightarrow !A; A^\perp$}
\RightLabel{$?R$}
\UnaryInfC{$\cdot \longrightarrow !A, ?(A^\perp)$}
\RightLabel{$\neg L$}
\UnaryInfC{$(?(A^\perp))^\perp \longrightarrow !A$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$B; B \longrightarrow B$}
\RightLabel{$\neg L$}
\UnaryInfC{$B; B, B^\perp \longrightarrow \cdot$}
\RightLabel{$!D$}
\UnaryInfC{$B; B^\perp \longrightarrow \cdot$}
\RightLabel{$?L$}
\UnaryInfC{$B; ?(B^\perp) \longrightarrow \cdot$}
\RightLabel{$!L$}
\UnaryInfC{$?(B^\perp), !B \longrightarrow \cdot$}
\RightLabel{$\neg R$}
\UnaryInfC{$?(B^\perp) \longrightarrow (!B)^\perp$}
\RightLabel{$\otimes R$}
\BinaryInfC{$?(B^\perp), (?(A^\perp))^\perp \longrightarrow !A \otimes (!B)^\perp$}
\RightLabel{$\otimes L$}
\UnaryInfC{$?(B^\perp) \otimes (?(A^\perp))^\perp \longrightarrow !A \otimes (!B)^\perp$}
\RightLabel{$\neg L$}
\UnaryInfC{$(!A \otimes (!B)^\perp)^\perp, ?(B^\perp) \otimes (?(A^\perp))^\perp \longrightarrow \cdot$}
\RightLabel{$\neg R$}
\UnaryInfC{$(!A \otimes (!B)^\perp)^\perp \longrightarrow (?(B^\perp) \otimes (?(A^\perp))^\perp)^\perp$}
\end{prooftree}
$$

Below we show the representation of the first derivation and the check that it is linear, the representation of the second derivation and its linearity check, and then the Elf query that applies the admissibility algorithm to these to deductions. There are 6768 different computation sequences of the non-deterministic admissibility algorithm, many of them leading to distinct cut-free derivations (they differ by permutations of successive inference rules). On a Dec Alpha it takes the current Elf implementation 221 seconds wall-clock time to enumerate all solutions. We leave it as an exercise to the diligent reader to translate the first solution to a sequent derivation in ordinary notation.

```
- top ();
Using: admit.elf linear.elf elim.elf
Solving for: ad ad! ad? linp linn lin lin2 elim
?- {A:o} {B:o}
lin2
([n:neg (! (perp (A times (perp B))))]
   [p:pos (perp ((! A) times (perp (! B))))]
   (perpr ([n1:neg ((! A) times (perp (! B)))]
            timesl ([n4:neg (! A)]
                     [n5:neg (perp (! B))]
                     perpl ([p6:pos (! B)]
                             !l ([n7!:neg! (perp (A times (perp B)))]
                                  !l ([n8!:neg! A]
                                       !r ([p9:pos B]
                                            !d ([n10:neg A]
                                                 !d ([n11:neg (perp (A times (perp B)))]
                                                      perpl ([p12:pos (A times (perp B))]
                                                              timesr ([p13:pos A] axiom n10 p13)
                                                              ([p14:pos (perp B)]
                                                                 perpr ([n15:neg B] axiom n15 p9)
                                                                 p14)
                                                              p12)
                                                      n11)
                                                 n7!)
                                            n8!)
                                       p6)
                                  n4)
                             n)
```

```
                       n5)
             n1)
      p)).

Solving...
solved
yes
?- {A:o} {B:o}
lin2
([n:neg (perp ((! A) times (perp (! B))))]
   [p:pos (perp ((? (perp B)) times (perp (? (perp A)))))]
   perpr ([n1:neg ((? (perp B)) times (perp (? (perp A))))]
            perpl ([p2:pos ((! A) times (perp (! B)))]
                     timesl ([n3:neg (? (perp B))]
                               [n4:neg (perp (? (perp A)))]
                               timesr ([p5:pos (! A)]
                                         perpl ([p6:pos (? (perp A))]
                                                  ?r ([p7?:pos? (perp A)]
                                                        !r ([p8:pos A]
                                                              ?d ([p9:pos (perp A)]
                                                                    perpr ([n10:neg A] axiom n10 p8)
                                                                    p9)
                                                              p7?)
                                                        p5)
                                                  p6)
                                         n4)
                               ([p11:pos (perp (! B))]
                                  perpr ([n12:neg (! B)]
                                           !l ([n13!:neg! B]
                                                 ?l ([n14:neg (perp B)]
                                                       !d ([n15:neg B]
                                                             perpl ([p16:pos B] axiom n15 p16)
                                                             n14)
                                                       n13!)
                                                 n3)
                                           n12)
                                  p11)
                               p2)
                     n1)
            n)
   p).

Solving...
solved
yes
?- {A:o} {B:o}
{N:neg (! (perp (A times (perp B))))}
{P:pos (perp ((? (perp B)) times (perp (? (perp A)))))}
ad (perp ((! A) times (perp (! B))))
([p:pos (perp ((! A) times (perp (! B))))]
   (perpr ([n1:neg ((! A) times (perp (! B)))]
             timesl ([n4:neg (! A)]
                       [n5:neg (perp (! B))]
                       perpl ([p6:pos (! B)]
                                !l ([n7!:neg! (perp (A times (perp B)))]
                                      !l ([n8!:neg! A]
                                            !r ([p9:pos B]
                                                  !d ([n10:neg A]
                                                        !d ([n11:neg (perp (A times (perp B)))]
                                                              perpl ([p12:pos (A times (perp B))]
                                                                       timesr ([p13:pos A] axiom n10 p13)
                                                                       ([p14:pos (perp B)]
                                                                          perpr ([n15:neg B] axiom n15 p9)
                                                                          p14)
                                                                       p12)
```

```
                                                             n11)
                                                        n7!)
                                                   n8!)
                                              p6)
                                         n4)
                                    N)
                            n5)
                   n1)
         p))
([n:neg (perp ((! A) times (perp (! B))))]
   perpr ([n1:neg ((? (perp B)) times (perp (? (perp A))))]
            perpl ([p2:pos ((! A) times (perp (! B)))]
                     timesl ([n3:neg (? (perp B))]
                               [n4:neg (perp (? (perp A)))]
                               timesr ([p5:pos (! A)]
                                         perpl ([p6:pos (? (perp A))]
                                                  ?r ([p7?:pos? (perp A)]
                                                        !r ([p8:pos A]
                                                              ?d ([p9:pos (perp A)]
                                                                    perpr ([n10:neg A] axiom n10 p8)
                                                                    p9)
                                                              p7?)
                                                        p5)
                                                  p6)
                                         n4)
                               ([p11:pos (perp (! B))]
                                  perpr ([n12:neg (! B)]
                                           !l ([n13!:neg! B]
                                                 ?l ([n14:neg (perp B)]
                                                       !d ([n15:neg B]
                                                             perpl ([p16:pos B] axiom n15 p16)
                                                             n14)
                                                       n13!)
                                                 n3)
                                           n12)
                                  p11)
                               p2)
                     n1)
            n)
   P)
(F A B N P).
Solving...

F =
   [A:o] [B:o] [N:neg (! (perp (A times perp B)))]
      [P:pos (perp (? (perp B) times perp (? (perp A))))]
      perpr
         ([n1:neg (? (perp B) times perp (? (perp A)))]
             timesl
                ([n11:neg (? (perp B))] [n2:neg (perp (? (perp A)))]
                    !l ([n1!:neg! (perp (A times perp B))]
                           perpl
                              ([p1:pos (? (perp A))]
                                  ?r ([p1?:pos? (perp A)]

?l ([n111:neg (perp B)]
       perpl
          ([p11:pos B]
              !d ([n1111:neg (perp (A times perp B))]
                     perpl
                        ([p1111:pos (A times perp B)]
                            timesr
                               ([p11111:pos A]
                                   ?d ([p9:pos (perp A)]
                                          perpr ([n10:neg A] axiom n10 p11111)
```

```
                                             p9)
                                      p1?)
                               ([p2:pos (perp B)]
                                   perpr ([n11111:neg B] axiom n11111 p11) p2)
                               p1111)
                        n1111)
                 n1!)
          n111)
   n11)
                                     p1)
                              n2)
                       N)
                n1)
         P.

yes
```

A.6 Sequent Calculus LV^+ with Cuts

```
%file lv-cut.elf

@ : type.         % Token (for derivations with Cuts)

%% Axioms

axiom^ : (neg A -> pos A -> @).

%% Multiplicative Connectives

timesr^ : (pos A -> @)
          -> (pos B -> @)
          -> (pos (A times B) -> @).

timesl^ : (neg A -> neg B -> @)
          -> (neg (A times B) -> @).

oner^ : (pos one -> @).

onel^ : @
        -> (neg one -> @).

%% Additive Connectives

andr^ : (pos A -> @) -> (pos B -> @)
        -> (pos (A and B) -> @).

andl1^ : (neg A -> @)
         -> (neg (A and B) -> @).

andl2^ : (neg B -> @)
         -> (neg (A and B) -> @).

topr^ : (pos (top) -> @).
```

```
% no topl

%% Involution

perpr^ : (neg A -> @)
         -> (pos (perp A) -> @).

perpl^ : (pos A -> @)
         -> (neg (perp A) -> @).

% Quantifier

forallr^ : ({a:i} pos (A a) -> @)
           -> (pos (forall A) -> @).

foralll^ : {T:i} (neg (A T) -> @)
           -> (neg (forall A) -> @).

%% Exponentials

!r^ : (pos A -> @)
      -> (pos (! A) -> @).

!l^ : (neg! A -> @)
      -> (neg (! A) -> @).

!d^ : (neg A -> @)
      -> (neg! A -> @).

?r^ : (pos? A -> @)
      -> (pos (? A) -> @).

?l^ : (neg A -> @)
      -> (neg (? A) -> @).

?d^ : (pos A -> @)
      -> (pos? A -> @).

%% Cuts

cut^ : (pos A -> @)
       -> (neg A -> @)
       -> @.

cut!^ : (pos A -> @)
        -> (neg! A -> @)
        -> @.

cut?^ : (pos? A -> @)
        -> (neg A -> @)
        -> @.
```

A.7 Cut Elimination for LV^+

```
%file elim.elf

elim : @ -> # -> type.

%mode -elim +D^ -D
%lex {D^}

ce_axiom  : elim (axiom^ N P) (axiom N P).

ce_timesr : elim (timesr^ D1^ D2^ P) (timesr D1 D2 P)
             <- ({p1} elim (D1^ p1) (D1 p1))
             <- ({p2} elim (D2^ p2) (D2 p2)).

ce_timesl : elim (timesl^ D1^ N) (timesl D1 N)
             <- ({n1} {n2} elim (D1^ n1 n2) (D1 n1 n2)).

ce_oner   : elim (oner^ P) (oner P).

ce_onel   : elim (onel^ D1^ N) (onel D1 N)
             <- elim D1^ D1.

ce_andr   : elim (andr^ D1^ D2^ P) (andr D1 D2 P)
             <- ({p1} elim (D1^ p1) (D1 p1))
             <- ({p2} elim (D2^ p2) (D2 p2)).

ce_andl1  : elim (andl1^ D1^ N) (andl1 D1 N)
             <- ({n1} elim (D1^ n1) (D1 n1)).

ce_andl2  : elim (andl2^ D2^ N) (andl2 D2 N)
             <- ({n2} elim (D2^ n2) (D2 n2)).

ce_topr   : elim (topr^ P) (topr P).

ce_perpr  : elim (perpr^ D1^ P) (perpr D1 P)
             <- ({n1} elim (D1^ n1) (D1 n1)).

ce_perpl  : elim (perpl^ D1^ N) (perpl D1 N)
             <- ({p1} elim (D1^ p1) (D1 p1)).

ce_forallr: elim (forallr^ D1^ P) (forallr D1 P)
             <- ({a:i} {p1} elim (D1^ a p1) (D1 a p1)).

ce_foralll: elim (foralll^ T D1^ N) (foralll T D1 N)
             <- ({n1} elim (D1^ n1) (D1 n1)).

ce_!r     : elim (!r^ D1!^ P) (!r D1! P)
             <- ({p1} elim (D1!^ p1) (D1! p1)).

ce_!l     : elim (!l^ D1^ N) (!l D1 N)
             <- ({n1!} elim (D1^ n1!) (D1 n1!)).
```

```
ce_!d     : elim (!d^ D1^ N!) (!d D1 N!)
             <- ({n1} elim (D1^ n1) (D1 n1)).

ce_?r     : elim (?r^ D1^ P) (?r D1 P)
             <- ({p1?} elim (D1^ p1?) (D1 p1?)).

ce_?l     : elim (?l^ D1!^ N) (?l D1! N)
             <- ({n1} elim (D1!^ n1) (D1! n1)).

ce_?d     : elim (?d^ D1^ P?) (?d D1 P?)
             <- ({p1} elim (D1^ p1) (D1 p1)).

ce_cut    : elim (cut^ D1^ D2^) F
             <- ({p} elim (D1^ p) (D1 p))
             <- ({n} elim (D2^ n) (D2 n))
             <- ad A D1 D2 F.

ce_cut!   : elim (cut!^ D1!^ D2^) F
             <- ({p} elim (D1!^ p) (D1! p))
             <- ({n!} elim (D2^ n!) (D2 n!))
             <- ad! A D1! D2 F.

ce_cut?   : elim (cut?^ D1^ D2!^) F
             <- ({p?} elim (D1^ p?) (D1 p?))
             <- ({n} elim (D2!^ n) (D2! n))
             <- ad? A D1 D2! F.
```

References

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

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

[Gir93] Jean-Yves Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[GP] Didier Galmiche and Guy Perrier. On proof normalization in linear logic. *Theoretical Computer Science*. To appear. Available as Technical Report CRIN 94-R-113, Nancy, France.

[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.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[Hod94] Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[MPP92] Dale Miller, Gordon Plotkin, and David Pym. A relevant analysis of natural deduction. Talk given at the Workshop on Logical Frameworks, Båstad, Sweden, May 1992.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[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.

[Wal90] Lincoln A. Wallen. *Automated Proof Search in Non-Classical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* MIT Press, 1990.